Jorge Miguel Martins Matosa

Licenciatura em Ciências da
Engenharia Eletrotécnica e de Computadores

Otimização de processos em empresas eletrotécnicas

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: Tiago Cardoso, Professor Doutor,
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa

Júri:

Presidente: Professora Doutora Maria Helena Fino, FCT/UNL

Arguente: Professor Doutor João Rosas, FCT/UNL

Vogal: Professor Doutor Tiago Cardoso, FCT/UNL

Março, 2018

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Otimização de processos em empresas eletrotécnicas

Gostaria de dedicar esta dissertação, primeiramente, aos meus pais, irmã e avó que sempre me deram todo o apoio e suporte em todas as matérias da minha vida e, acima de tudo, deram-me força para finalizar este grande ciclo da minha vida que é o mestrado e a licenciatura.

Queria também agradecer aos meus colegas e amigos de curso porque sempre me ajudaram quando foi necessário, assim como eu os ajudei a eles.

O trabalho de aconselhamento do meu orientador, Tiago Cardoso, na realização desta dissertação também foi muito importante para manter o trabalho na linha correta.

Por fim, queria agradecer à minha namorada e aos meus amigos por me terem suportado e me terem feito ter sempre uma atitude positiva na vida.

Resumo

No contexto empresarial atual verifica-se alguma desorganização da gestão interna de processos, algo que dificulta bastante a eficiência de produção de trabalho.

Principalmente nas grandes empresas, existem múltiplas ferramentas complexas que apenas são utilizadas com pequenos propósitos e, como tal, pretende-se estudar uma solução que permite resolver o problema com recurso a uma só solução.

Pretende-se, nesta dissertação, estudar o método de trabalho mais utilizado nas empresas de desenvolvimento de software, assim como as ferramentas mais utilizadas e, por fim, as *frameworks* atuais que poderão ajudar a resolver o problema de excessiva complexidade atualmente presente no mercado.

Será também apresentada uma proposta conceptual que visa solucionar pequenos problemas estudados.

Palavras-chave: *frameworks,* desenvolvimento de software, organização de processos

Abstract

Today, in the corporative world, we can observe the lack of organization in the companies' internal processes, something that disturbs the efficiency of the work produced.

Mainly in big companies, there are too many complex tools used for, sometimes, simple tasks and this thesis' objective is to study a solution which could potentially solve part of this problem using only one platform.

The common work method of the software developing companies will also be studied, as well as some of the most used tools in these companies and the trending frameworks between software developers of today, frameworks which will help reduce this excessive complexity present in the market.

It will be also be presented a conceptual proposal to a solution of some small problems inside the studied subject.

Keywords: frameworks, software development, processes organization

Índice

Lista de Tabelas

Lista de Figuras

Lista de Acrónimos

Web-based Termo utilizado para algo que é construído com o objetivo de ser apenas utilizado através de um navegador na internet e não instalado no dispositivo.

Framework Quando este termo é utilizado, refere-se a um software que dispõe de funcionalidades pré-configuradas que permitem que o utilizador as consiga utilizar para construir algo com base nessas funcionalidades.

Tickets Pedidos realizados via humana cujo objetivo é chegarem a outros operadores que os terá de resolver.

User Friendly Algo que é criado com o objetivo de que a interação com o utilizador seja muito fácil e acessível.

IT Termo utilizado para tecnologias de informação (Information Technologies).

Sprints Designação dos períodos de programação extrema no tipo Agile de desenvolvimento de software.

Big Data Conjunto de dados tão volumoso que as aplicações tradicionais de processamento de dados não têm capacidade para tratar.

Changes Número de identificação de resolução de pedidos em desenvolvimento de software.

Defects Defeito de software reportado por um utilizador/automatismo.

Scrum *Framework* para gestão de trabalho com ênfase no desenvolvimento de software. Esta *framework* é desenhada para equipas de três a seis pessoas.

Bugs Problemas reportados em software.

Runtime Período referente à execução de um programa.

API Conjunto de sub-rotinas, protocolos e ferramentas com o objetivo de existir comunicação entre programas.

SPA Definição referente ao método aplicacional do Angular que se refere a uma *Single Page Application*, isto é, toda a aplicação está contida apenas numa página e não em várias.

DOM É uma interface que utilizada em todas as plataformas que trata os documentos HTML, XHTML e XML como uma estrutura em árvore onde cada nó é um objeto que representa parte do documento.

1

1 Introdução

O trabalho proposto nesta dissertação está inserido no campo de desenvolvimento de software para empresas.

Atualmente, principalmente em grandes empresas, existem múltiplas ferramentas para realizar determinadas tarefas, tarefas estas tão pequenas que não seria necessária uma ferramenta muito complexa para tal fim. Estas múltiplas plataformas, por vezes, até acabam por sobrepor funcionalidades e é aqui que o desperdício de recursos realmente começa.

Este problema existe, principalmente, pela falta de planeamento a longo prazo. Na maior parte das ferramentas seguidamente estudadas, denota-se que as funcionalidades foram sempre acrescentadas por cima duma base previamente planeada. Uma vez que não foram planeadas, tornaram uma tarefa básica numa tarefa não tão fácil para o utilizador comum.

No contexto empresarial atual, o crescente número de dados com que cada empresa tem de lidar no século XXI implica que a organização empresarial seja mais eficiente para conseguir tirar maior proveito deste facto e focar-se na diversidade destes dados e não na quantidade (Davenport & Dyché, 2013). É neste contexto que esta dissertação está inserida: simplificar tarefas e mostrar, com recurso a ferramentas atuais, como se pode abordar a comunicação entre os trabalhadores sem que seja necessário gastar muitos recursos.

Com a crescente utilização do Javascript pelas diversas *frameworks* de desenvolvimento web (Graziotin & Abrahamsson, 2013), torna-se cada vez mais fácil criar plataformas deste tipo e centralizar a informação a aceder pelos colaboradores na empresa sem que haja necessidade de estar constantemente a atualizar software local nos computadores dos mesmos. A facilidade/capacidade de criar ferramentas cada vez mais robustas permitem seguir esta abordagem duma maneira que outrora não seria possível, uma vez que o mercado não disponibilizava recursos para seguir esta via. O facto da maior parte das grandes *frameworks* utilizadas serem *Opensource* é também um fator decisivo para a escolha das empresas quando decidem seguir este caminho.

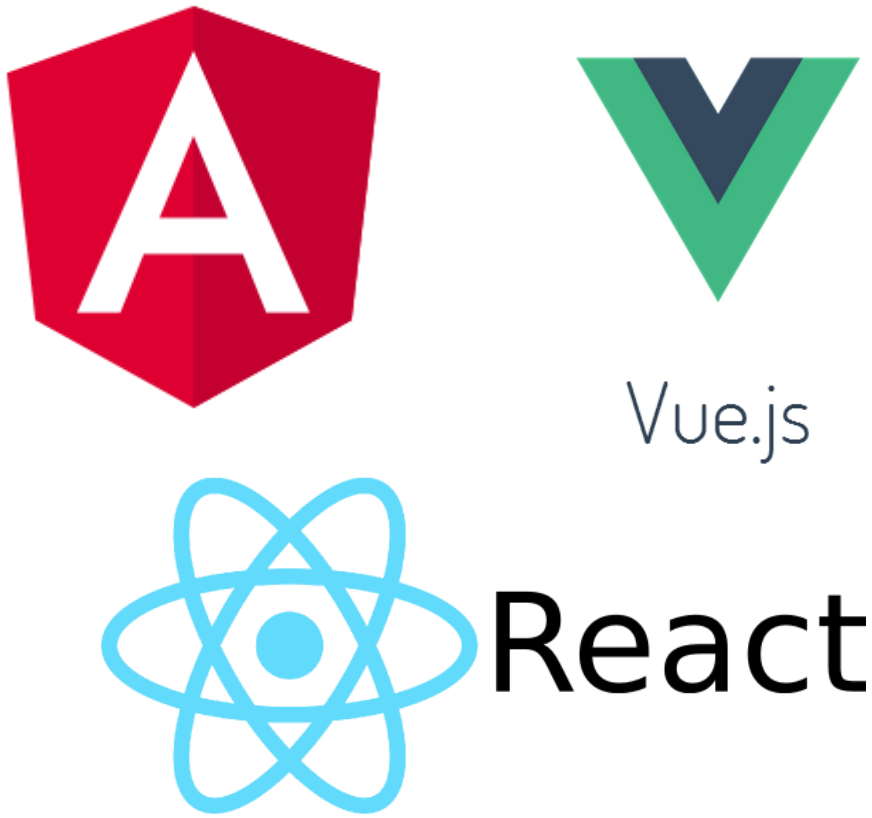

Figura 1.1 - Frameworks de desenvolvimento web mais utilizadas em 2018

Propõe-se então a realização de uma proposta conceptual, com recurso a *frameworks* existentes no mercado, que consiga proporcionar variadas funcionalidades importantes num só local. A abordagem a este problema irá seguir o caminho da modularidade, simplicidade e, acima de tudo a possibilidade de poder ser atualizada no futuro. O objetivo desta dissertação passa por abrir uma janela para a organização interna, desde pequena a grande escala, nas empresas, assim como utilizar esta organização para conseguir analisar todo o tipo de dados a que as empresas têm acesso da forma mais eficiente possível.

1.1 Motivação

O crescente desenvolvimento das ferramentas disponíveis para os programadores, assim como a facilidade de utilização e crescimento de funcionalidades, permite que cada vez mais pessoas se aventurem nesta área.

Aliando a necessidade de organização e restruturação das empresas para um mercado muito diferente do que era há 20 anos atrás, cada vez se vê mais novas ideias a surgir, ideias estas capazes de revolucionar o mundo.

O situação monetária das instituições quando abordam novos caminhos também é um fator crucial no desenvolvimento das mesmas e, como muitas das ferramentas fornecidas por empresas como a Google e o Facebook são grátis e *opensource*, dá garantias às empresas que estão a escolher um produto viável e com futuro.

Uma vez que a área do desenvolvimento-web está cada vez mais em crescimento, é um assunto bom para explorar e tirar partido da grande comunidade presente neste ramo para tornar a vida das pessoas e das empresas em algo cada vez mais fácil, produtivo e menos stressante.

Tendo como base algum trabalho efetuado na NOS (uma das maiores empresas de telecomunicações do país), verificou-se que a organização das ferramentas e dos trabalhadores é um fator crucial para que o bem-estar dos trabalhadores e produtividade da empresa estejam no melhor caminho, não sendo isso que se verifica na maior parte das situações.

A motivação presente alia esta vontade de criar algo produtivo que possa, eventualmente, ser utilizado para o bem das pessoas e aprender um pouco sobre

o mundo destas ferramentas tão importantes que a comunidade de programadores tanto fala.

1.2 Organização da Tese

A organização da tese será feita em três componentes principais:

- Estudo das ferramentas existentes no mercado e que funcionalidades poderão ser incorporadas na aplicação;
- Projeto e planeamento da aplicação;
- Implementação da aplicação.

Também será feita uma validação dos da aplicação e serão tiradas conclusões sobre o trabalho desenvolvido.

2

2 Estado da Arte

2.1 Introdução

Esta dissertação irá incluir três bases fundamentais na sua construção, bases estas que permitirão uma desenvolvimento equilibrado da aplicação.

A primeira base é o desenvolvimento de software, área esta que permitiu a ideia fundamental para o tema da dissertação. Aqui serão estudadas quais as necessidades da mesma para uma otimização do desenvolvimento e como é que uma ferramenta como a proposta se poderá inserir no contexto do área. Será também explicado melhor como é que a metodologia *Agile* pode beneficiar o ambiente de produção de software.

De seguida, serão abordadas algumas aplicações que são utilizadas para a otimização de processos, serão realçados os prós e os contras e será discutido como é que a ferramenta podia ser mais *user friendly* ou, caso já o seja, será indicado o conjunto de fatores que permitem essa classificação.

Por fim, serão discutidas as ferramentas que poderão ser utilizadas para o desenvolvimento desta aplicação. Haverá um foco especial nas ferramentas mais atualizadas como Angular 2+, ReactJS e Vue.

2.2 Desenvolvimento de software

2.2.1 Definição

O processo de desenvolvimento de software trata-se de um conjunto de atividades e resultados que ajudam a produção de software, como por exemplo a análise de requisitos e a codificação (Soares, 2003).

Para este fim, existem metodologias predefinidas que a grande maioria das empresas seguem de forma a serem mais eficientes, metodologias estas que passam, geralmente, pelo seguinte:

- Especificação de software;
- Projeto e implementação de software;
- Validação de software;
- Evolução de software.

O primeiro passa pela definição das funcionalidades e requisitos do software, onde a comunicação com o cliente é um fator principal para atingir os resultados desejados. O segundo, como o nome indica, é a fase de desenvolvimento do software, utilizando esquemas e diagramas. Na terceira fase verifica-se se todos os requisitos desenvolvidos estão de acordo com o pedido e o quarto trata de fazer novos desenvolvimentos no projeto de forma a o deixar relevante para o futuro.

2.2.2 Estado atual

O modelo predominante para o desenvolvimento de software é o modelo sequencial (Pressman, 2011). Este foi o primeiro modelo a ser criado e é um modelo muito restrito devido à intransigência no cumprimento das tarefas de acordo com as fases propostas. A Figura 2.1 mostra a "cascata" que representa as fases de desenvolvimento neste modelo.

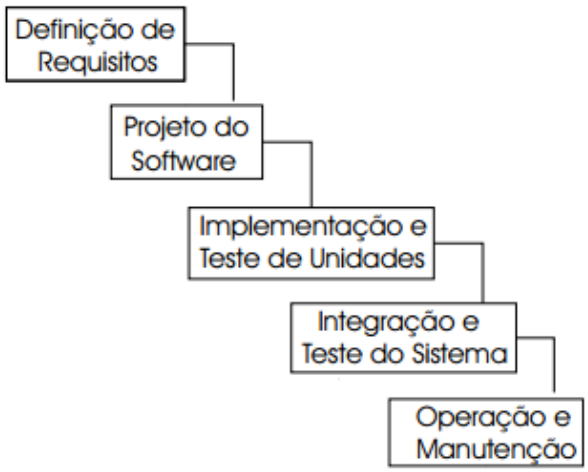

Figura 2.1 - Modelo Clássico

Uma vez que todas as etapas estão muito bem definidas, é muito difícil proceder a alterações de última hora. Quando os projetos são muito grandes e complexos as alterações podem demroar meses ou anos a acabar e podem já não ser tão relevantes como quando foram pensados.

Para a resolução deste problema, surgiram as metodologias ágeis como o ***Extreme Programming*** (Beck & Andres, 2004) e ***Scrum*** (Schwaber & Beedle, 2001)**.**

A primeira metodologia, à semelhança da segunda, tem como objetivo o desenvolvimento rápido. Esta foca-se em quatro pilares: comunicação, simplicidade, *feedback* e coragem (Beck & Andres, 2004). Este método trata-se, então, da utilização de equipas pequenas que vão desenvolver o software com requisitos pouco explícitos e que se vão modificando ao logo do tempo conforme a necessidade.

O outro método utilizado, ***Scrum***, é, possivelmente, a alternativa ao modelo clássico mais utilizada (Schwaber & Beedle, 2001). A caracterização deste modelo é muito semelhante à do anterior, mas a dimensão temporal de desenvolvimento diferente bastante, isto é, o desenvolvimento é feito com base em *sprints* de trinta dias, onde as equipas são constituídas por projetistas, programadores, engenheiros e gestores de qualidades, num máximo de dez pessoas. Estas equipas trabalham sempre sob os requisitos definidos no início dos sprints e existem sempre reuniões diárias para reportar o progresso e definir novos objetivos.

2.3 Software utilizado

Com base no que foi referido sobre os processos de desenvolvimento de software em empresas, as mesmas necessitam de ferramentas sob as quais consigam trabalhar e comunicar decisões/problemas.

Nesta secção serão dados exemplos de aplicações que são utilizadas nas empresas de desenvolvimento de software e para que efeito são utilizadas.

2.3.1 HP Service Manager (HPSM)

O HPSM (Figura 2.2) é uma ferramenta de **Central de Serviços** (*Service Desk)* onde é possível gerir os problemas da empresa através da abertura de pedidos (tickets), pedidos estes que vão para os responsáveis para a resolução dos mesmos.

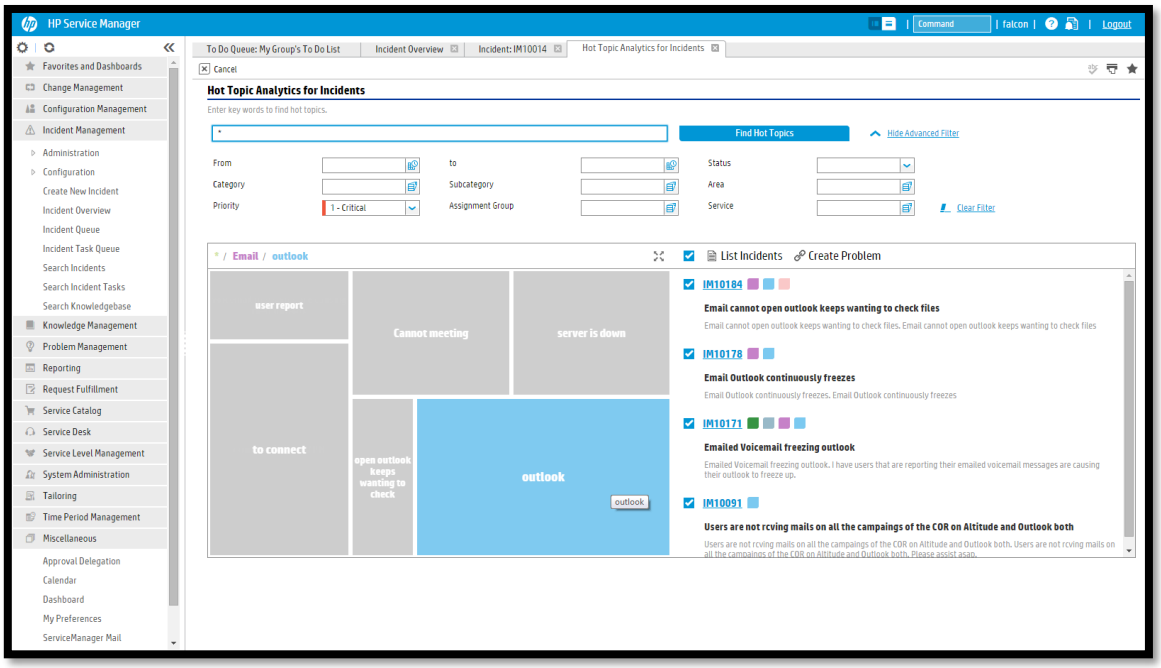

Figura 2.2 - HPSM

Esta ferramenta permite centralizar todos os aspetos de resolução de serviços e oferece a capacidade de gestão de tickets através dos utilizadores, podendo estes criá-los e fechá-los conforme o necessário.

Pode-se encontrar as seguintes funcionalidades no HPSM:

- Central de serviços completo;

- Inteligência para *Big Data;*
- Gestão de *changes* em *Agile* e automatismos;
- Suporte via telefone ou tablet.

Um dos grandes problemas desta plataforma é o facto de ser muito pouco intuitiva. Para se abrir um ticket é necessário o utilizador já conhecer muito bem a plataforma e, a grande maioria das vezes, é uma tarefa muito entediante porque a abertura de um ticket é muito complexa e requer muito tempo ao utilizador.

2.3.2 HP Quality Center (HPQC)

Esta ferramenta tem como objetivo gerir a qualidade de software.

Para tal objetivo, o HPQC (Figura 2.3) fornece alguns serviços como gestão de pedidos, plano de qualidade, gestão de testes de software, teste de processos de negócio e **gestão de *defects***.

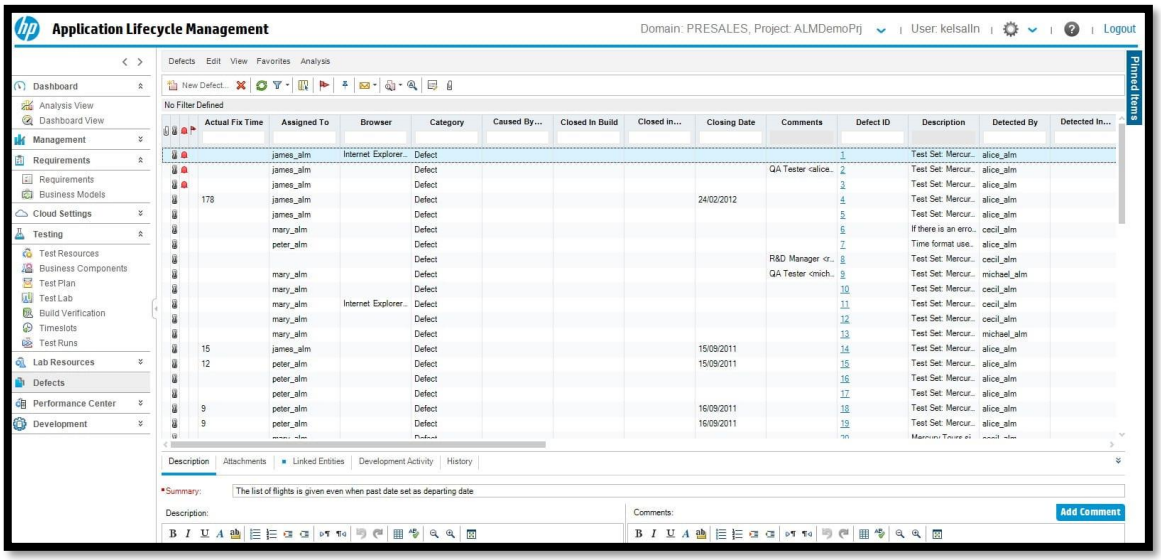

Figura 2.3 – HPQC

A sua principal função é evitar problemas de software de forma a que este possa chegar a produção com o mínimo de erros possíveis.

É possível controlar todos estes processos através de uma plataforma *web-based* e também é assim permitido que todos os intervenientes no processo de produção de software estejam sincronizados com os problemas encontrados.

O seu foco principal passa pela gestão de requerimentos, mas as seguintes funcionalidades também se encontram presentes:

- Colaboração entre programadores;
- Testes de *Agile*;
- Testes de automação;
- Gestão de testes e de *defects*;
- Sincronização empresarial.

Este tipo de software permite que os utilizadores padronizem e automatizem todos os processos envolvidos na gestão de requerimentos, testes e componentes de negócio, melhorando a velocidade de trabalho e mantendo uma consistência simultaneamente.

2.3.3 Jira

O Jira (Figura 2.4) é uma ferramenta de planeamento e gestão de projetos onde é possível organizar variados tipos de tarefas de forma a promover um funcionamento eficiente dos trabalhados.

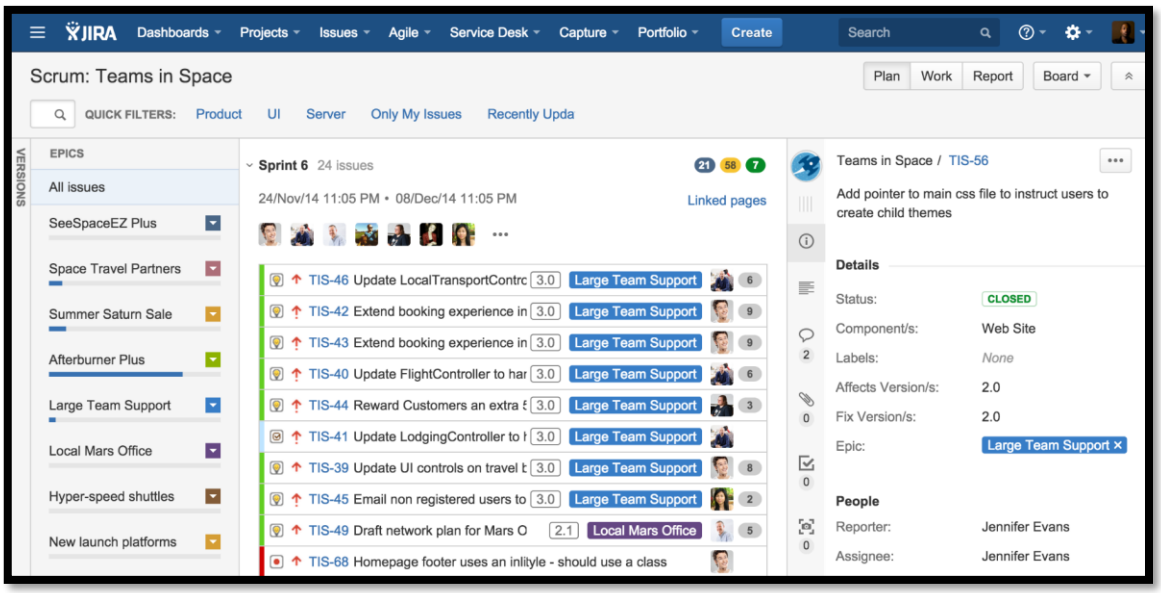

Figura 2.4 - Atlassian Jira

Aqui é permitido criar um fluxo de trabalho através da abertura de tarefas para/pelos trabalhadores, podendo estas tarefas ser organizadas por *Scrum boards*, *Kanban boards* e/ou *Agile reporting*.

Uma das principais vantagens deste tipo de software é a habilidade de poder criar fluxos de trabalho do género *Agile*, género este baseado em sprints de produção de trabalho e muito focado em tarefas de curto prazo de forma a conseguir reagir às adversidades e poder alterar as grandes tarefas a longo prazo conforme os problemas que vão surgindo.

Sendo este um dos três melhores softwares utilizados em 2017 (Glover et al., n.d.-a), tem, obviamente, muitas funcionalidades úteis para os utilizadores, como :

- Fluxos de trabalho customizáveis;
- Gestão de *bugs* e *defects;*
- Importação facilidade de outros sistemas;
- Interface móvel;
- Poderosa ferramenta de pesquisa e filtragem de conteúdos.

Devido à sua dimensão e otimização, esta é a ferramenta ideal para as empresas conseguirem gerir diferentes tipos de fluxo de trabalho da forma que acham mais necessária.

2.3.4 Oracle APEX

O APEX (Figura 2.5) é uma *framework* para criação de sites com base nas bases de dados da Oracle.

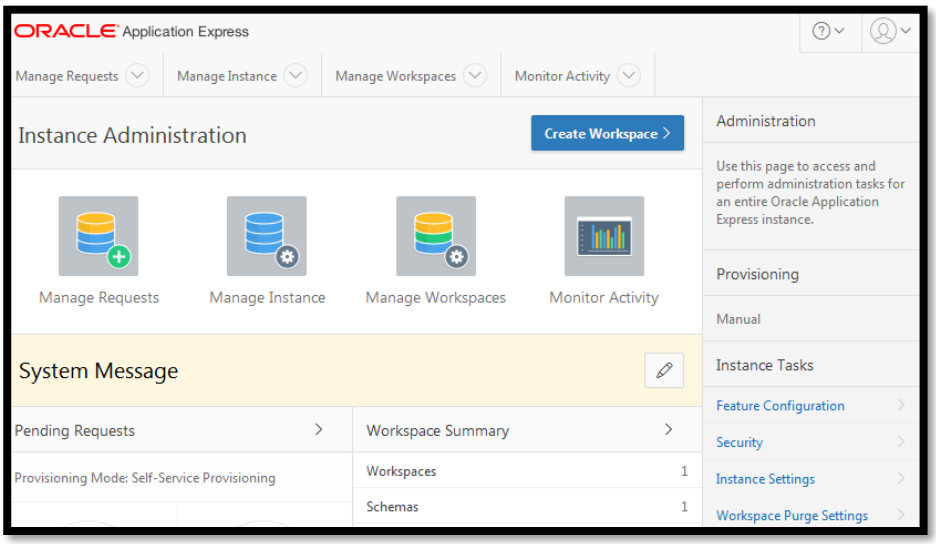

Figura 2.5 - Oracle Application Express

O APEX é criado sob uma base de dados e permite assim uma interação direta e simplificada com determinados dados que queiramos apresentar no site.

Esta plataforma permite desenhar sites com pouco conhecimento de *web-design* e é ideal para quem tem bastante conhecimento em SQL.

O APEX é principalmente usado por quem tem licenças de utilização das BD's da Oracle, uma vez que é grátis para quem as tem.

Um dos problemas da plataforma é que é pouco intuitiva e não permite obter plataformas muito complexas que saiam fora do padrão do APEX, sendo também a sua eficiência muito inferior a outras soluções já existentes no mercado.

2.3.5 Confluence

O Confluence (Figura 2.6) é um software de colaboração de equipas, que permite guardar, partilhar e trabalhar em variados tipos de informação nele inserido.

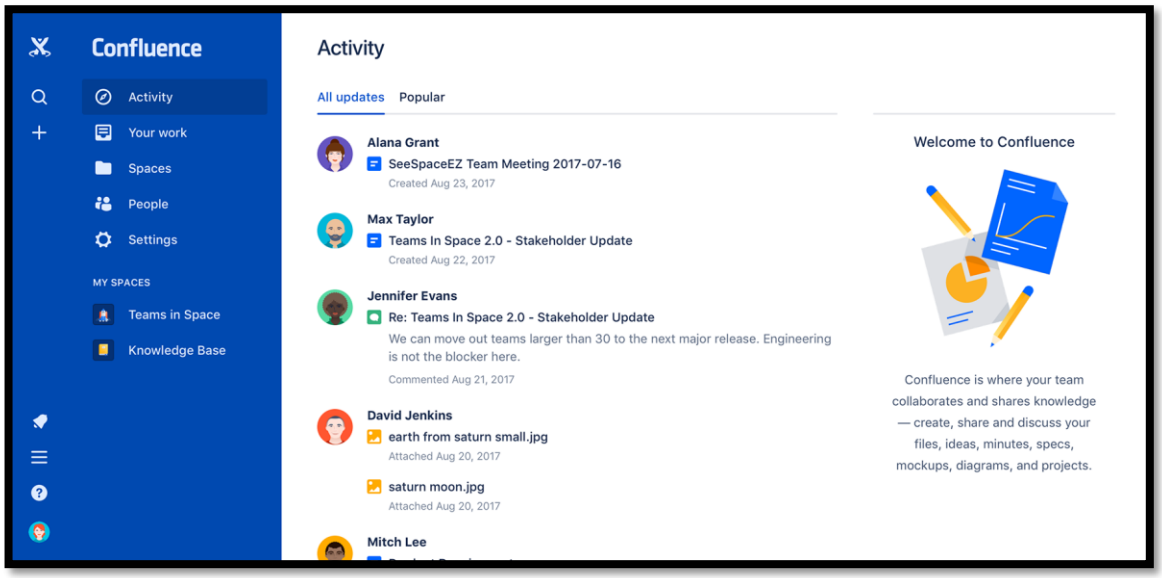

Figura 2.6 - Atlassian Confluence

Aqui é permitido centralizar toda a documentação que, outrora, estaria espalhada por variadas localizações, criando assim uma única localização onde é possível ao trabalhador encontrar o que deseja.

O Confluence tem as seguintes funcionalidades disponíveis:

- Colaboração de equipas;
- Desenvolvimento em *Agile;*
- Editor de texto;
- Base de dados;
- Integração com o Jira;
- Calendários;
- Notificações de tarefas.

À semelhança do Jira, a sua simplicidade torna-o numa boa solução de partilha de informação e é uma boa escolha se a necessidade da empresa/projeto for apenas essa.

2.3.6 Aqua ALM

O Aqua (Figura 2.7) é uma aplicação de gestão de ciclo de software com diferentes funcionalidades.

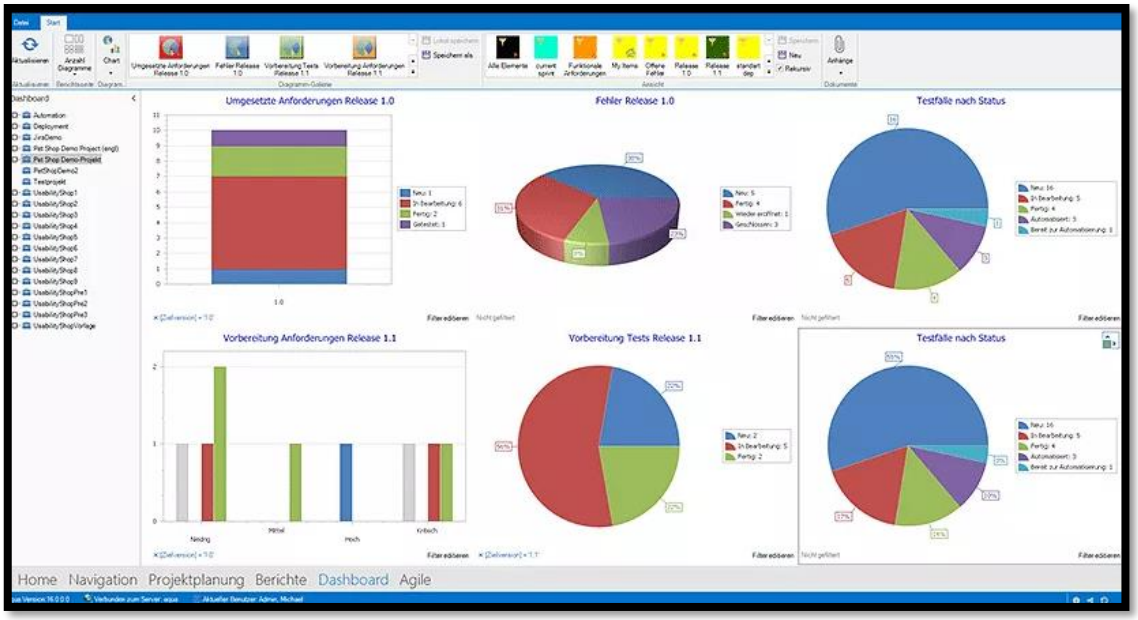

Figura 2.7 - Andagon Aqua ALM

O Aqua permite um desenvolvimento Agile dos produtos, gestão de testes automáticos, contém um Dashboard onde é possível publicar o progresso, permite fazer a gestão de projetos e também a gestão de *defects*.

Este produto também acaba por ser usado como repositório de informação em certas empresas e acaba por perder um pouco o propósito para o qual foi criado.

As suas principais funcionalidades passam por:

- Gestão de *defects* e *changes*;
- Gestão de testes;
- Especificação de testes.

2.3.7 Sharepoint

O Sharepoint (Figura 2.8) é uma plataforma da Microsoft que permite o desenvolvimento de aplicações Web como portais internos de empresas, repositórios de informação e conteúdos, assim como a sua gestão de documentos e também permite a criação de portais colaborativos.

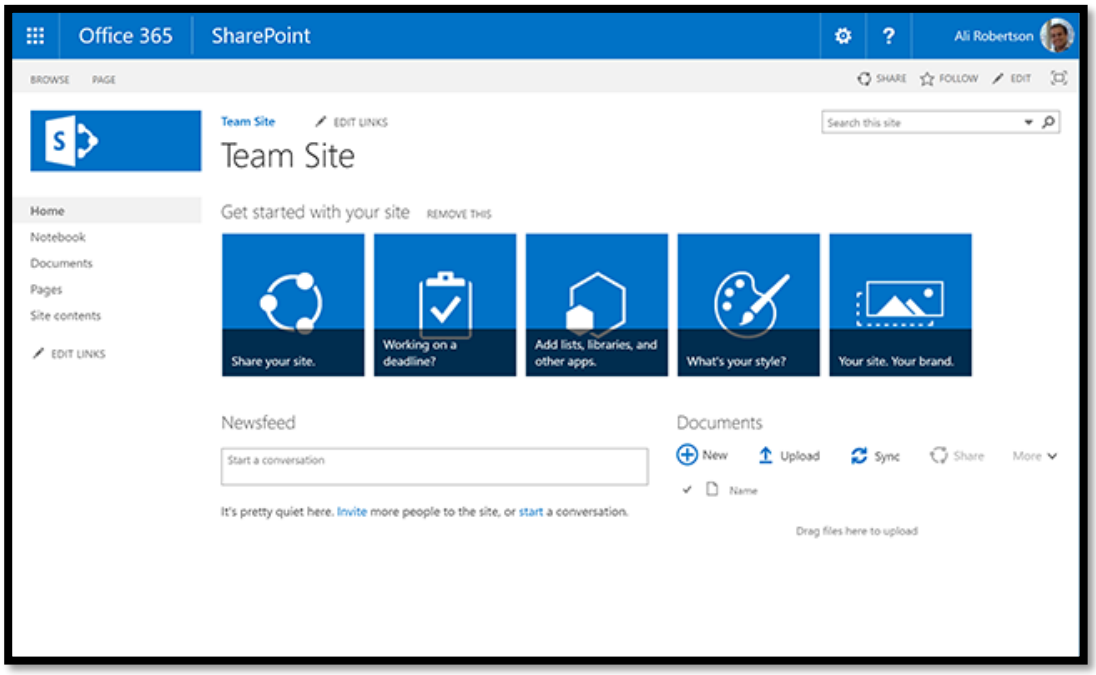

Figura 2.8 - Sharepoint

O Sharepoint nunca foi conhecido pela sua alta eficiência, pelo que tem muitos bugs, bugs estes que tornam, por vezes, a utilização do Sharepoint pouco agradável para o utilizador.

A gestão de documentos é a sua principal função, sendo que a maior parte das aplicações desenvolvidas em Sharepoint têm este fim.

Definindo-se como uma das principais plataformas de colaboração no mercado, o Sharepoint tem disponível o seguinte:

- Bibliotecas;
- Conexões encriptadas;
- Identificação de conteúdos sensíveis;
- Gestão de direitos de informação;
- Suporte para ficheiros de tamanho elevado;
- Suporte móvel;
- Inteligência de Negócio.

Sendo uma ferramenta da Microsoft, o Sharepoint é utilizado em diversas empresas.

2.3.8 SAP ERP (Enterprise Resource Planning)

O SAP ERP (Figura 2.9) é um software de total planeamento empresarial. É um produto vastamente utilizado por grandes empresas e o seu propósito é aumentar a eficiência de decisões através do tratamento de dados obtidos por diversas fontes.

Figura 2.9 - SAP ERP

Este sistema consegue centralizar grande parte da informação da empresa de forma a conseguir verificar que causas e efeitos podem afetar cada ramo da empresa.

Esta divisão da empresa é dada por módulos, sendo cada módulo uma área específica. Cada um destes módulos trata da gestão de variados processos de negócios e baseia-se nas práticas do dia-a-dia das empresas.

As funcionalidades padrão deste software são:

- Identificação de custos de fornecedores;
- Gestão de finanças;
- Planeamento de produto;
- Gestão de Recursos Humanos.

Podendo trazer os seguintes benefícios a uma empresa:

- Eliminar processos totalmente manuais;
- Otimizar o fluxo da informação e a qualidade da mesma dentro da organização (eficiência);
- Otimizar o processo de tomada de decisão;
- Reduzir os limites de tempo de resposta ao mercado;
- Incorporar melhores práticas (codificadas no ERP) aos processos internos da empresa;
- Reduzir o tempo dos processos gerenciais;
- Reduzir a carga de trabalho, pois atividades repetitivas podem e devem ser automatizadas;
- Melhorar o controlo das operações da empresa;

2.3.9 Conclusões e comparações

Na Tabela 2.1 podem-se observar algumas características mais objetivas relativamente aos produtos estudados, características estas mais focadas à escolha do produto por parte das empresas.

Pode-se notar que a maior parte destes estão disponíveis em todas as plataformas de utilização, podendo excluir este critério como escolha.

A experiência do consumidor é um dos principais fatores a ter em conta. Tem-se que o SAP ERP é o software com melhor experiência do utilizador e o HPSM o que tem a pior.

O fator financeiro é, a par da experiência do utilizador, um dos fatores mais relevantes. O SAP, apesar de ter a melhor experiência, é também, de todos, o software mais caro.

Todos estes dados foram retirados de (Glover et al., n.d.-b).

Tabela 2.1 - Comparação de características do software estudado

Software	Experiência do consumidor (%)	Custo por 100 utilizadores	Integração com outro software	Plataformas de utilização	Empresas alvo
HPSM	84	5000$/mês	Sim	windows	médias e grandes
HPQC	99	4000$/licença	Sim	windows, android, mac, linux, iphone/ipad	médias e grandes
JIRA	99	450$/mês	Sim	windows, android, mac, linux, iphone/ipad, web-based	pequenas, médias e grandes
APEX	92	Grátis utilizando a Oracle Database	Apenas com bases de dados Oracle	web-based	pequenas, médias e grandes
Confluence	98	300$/Mês	Sim	windows, android, mac, linux, iphone/ipad, web-based	pequenas, médias e grandes
Sharepoint	97	2000$/mês	Sim	windows, android, mac, linux, iphone/ipad, web-based	pequenas, médias, grandes e freelancers
SAP ERP	100	50000$/licença	Sim	windows, android, mac, linux, iphone/ipad	pequenas, médias e grandes

Na Figura 2.10 pode-se ver representada a ligação entre os campos de ação de cada aplicação estudada, campos estes referidos em cada software, e até onde é que a utilização simultânea de duas aplicações pode causar uma sobreposição de funcionalidades.

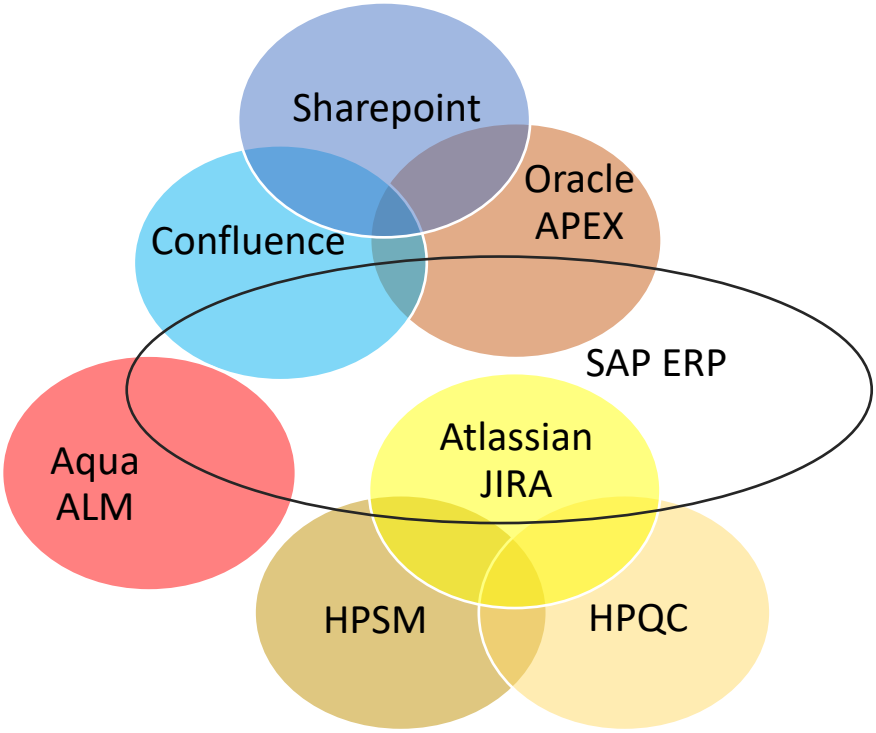

Figura 2.10 - Diagrama de sobreposição de funcionalidades

O SAP ERP, devido a ser o software mais trabalhado e à sua grande modularidade é, de todos, o software mais completo, daí a sua área de aplicação estar coincidente com quase todas as outras aplicações.

O Sharepoint é o software que se foca mais num nicho de mercado que nenhum dos outros softwares se foca, por isso fica fora do campo de ação do SAP.

Todos as outras ferramentas focam-se, também, no seu nicho de mercado mas, o SAP, com o número de módulos extremamente elevado, consegue praticamente cobrir as funcionalidades mais importantes.

A melhor solução para uma empresa, caso tivesse disponibilidade monetária, seria o SAP ERP.

2.4 *Frameworks* introduzidas no mercado

Para o desenvolvimento de aplicações *web-based*, existem quatro principais *frameworks*/bibliotecas para o efeito: **ReactJS, Vue.js, AngularJS** e **Angular 2+**.

Como se pode ver no gráfico abaixo, a popularidade de pesquisas no Google disputa-se entre Angular e React, dividindo-se angular pelas duas versões mencionadas.

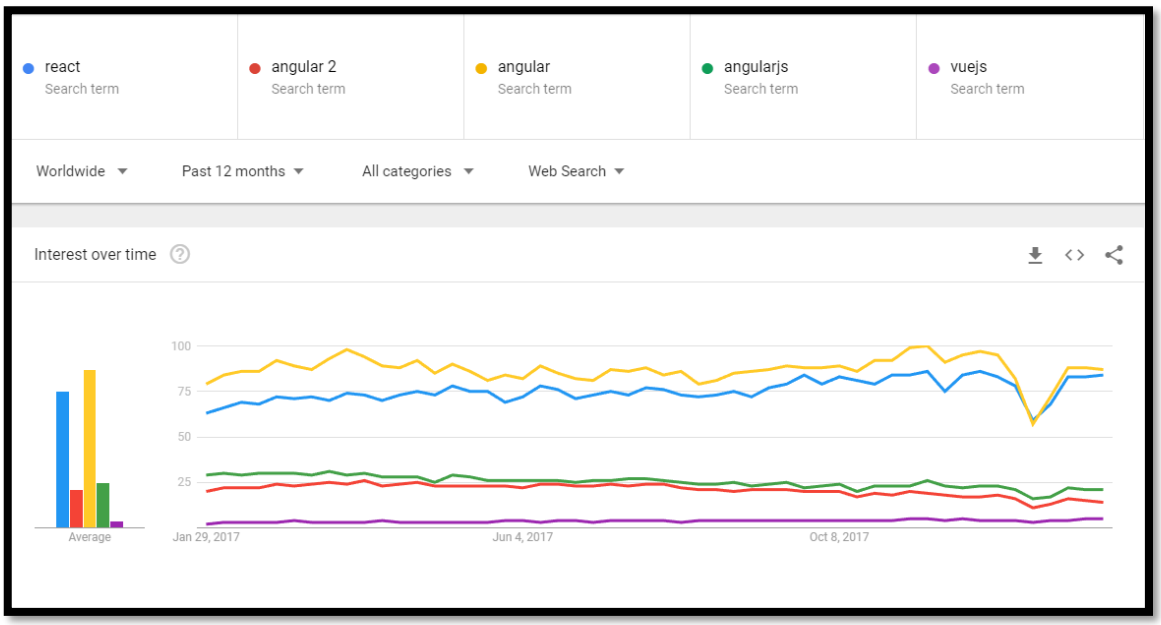

Figura 2.11 - Número de pesquisas por dia de cada *framework*

Nesta secção vão ser apresentadas as *frameworks* estudadas e o porquê de serem as mais utilizadas atualmente, à exceção de AngularJS, uma vez que é a versão anterior ao Angular 2.

2.4.1 Angular 2+

O Angular 2+ (atualmente na versão 5), é o sucessor do AngularJS. É uma *framework* mais robusta que o seu antecessor e permite construir aplicações web com alguma facilidade. Esta ferramenta combina *templates* declarativos, injeção de dependências, ferramentas "*end to end*" e, no geral, melhores práticas para resolução de problemas. Esta *framework* permite também construir aplicações móveis e de *desktop*.

Esta ferramenta é baseada em **Typescript** que é uma modificação glorificada do **Javascript**, isto é, é uma modificação que permite definir tipos de variáveis e que requer compilação antes do uso, ao contrário de Javascript normal.

Para a construção de aplicações, o Angular baseia-se em **nove** princípios fundamentais, sendo eles apresentados de seguida.

2.4.1.1 Arquitetura e Módulos

A arquitetura do Angular está apresentada na Figura 2.12 - Arquitetura de uma aplicação em Angular.

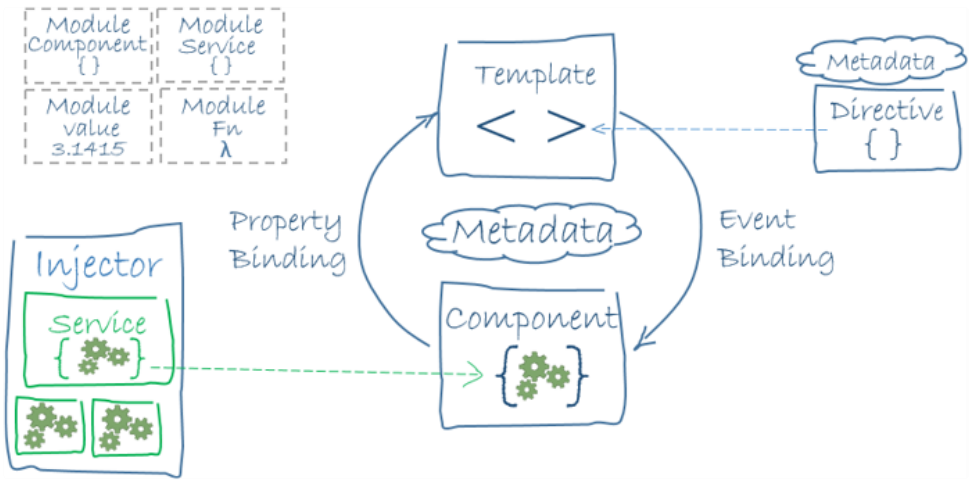

Figura 2.12 - Arquitetura de uma aplicação em Angular (Google, n.d.)

Os módulos são os principais trabalhadores da aplicação. Cada módulo vai ter código referente a zonas da páginas que se deve apresentar, sendo o número de módulos completamente à escolha do programador, sendo que uma aplicação pode apenas ter um componente.

Como principal funcionalidade temos os **componentes**. Estes vão conter a estrutura básica da aplicação, assim como alguma lógica necessária para o funcionamento da mesma.

Para efetuar a comunicação entre componentes existem os **serviços**. Estes podem ser injetados nos componentes ou noutros serviços e irão conter informação que será utilizada em vários locais na aplicação.

Para além disto, existem também classes normais de Javascript/Typescript e funções que poderão ser utilizadas.

As diretivas são instruções para alterar o dinamicamente o DOM (*Document Object Model)*, ou seja, o que nos é apresentado no browser. O angular já tem várias diretivas pré-definidas, mas pode-se também criar diretivas customizadas para uma interação mais detalhada com o utilizador.

2.4.1.2 Data Binding

O *Data Binding* é uma das principais funcionalidades do Angular. Com esta funcionalidade é permitido que haja troca de informação entre o *template* HTML e o Angular em si. Outrora seria necessário criar muita lógica de *push* e *pull* que tornaria muito difícil de perceber o que estava no código e também criaria muitos erros.

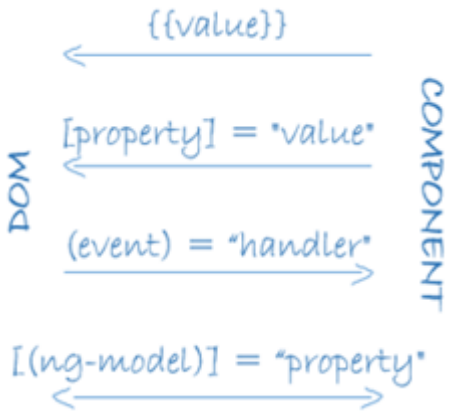

Figura 2.13 - Demonstração de *Data Binding* (Google, n.d.)

2.4.1.3 Formulários

Os formulários são também uma funcionalidade útil presente no Angular. Permite criar formulário baseados no *template* ou formulário reativos em que se consegue gerir melhor como a informação é tratada.

2.4.1.4 NgModules

As classes NgModule permitem declarar como um objetivo vai ser compilado e como criar um injentor em *runtime.* Esta vai identificar os componentes, diretivas e *pipes* de cada módulo e permite, assim, organizar a aplicação na sua raiz através das declarações.

2.4.1.5 Injeção de Dependências

A injeção de dependências, como o nome indica, é utilizada para evitar que seja necessário estar constantemente a declarar dependências em cada sítio que uma classe é declarada. Cada classe fica responsável pelas suas próprias dependências e, caso estas não sejam modificadas em mais nenhum lado, não é preciso declará-las em mais nenhum sítio que esta classe-mãe.

2.4.1.6 HttpClient

Esta funcionalidade é uma das mais importantes porque é a que permite a comunicação entre o *front-end* (Angular) e a ferramente de *back-end*.

Esta é uma *API* simplificada que é fornecida pelo Angular que constrói sobre um dos métodos de fazer pedidos HTTP (XMLHttpRequest).

2.4.1.7 Navegação e Roteamento

O angular tem um método de roteamento embutido que permite navegar entre páginas conforme o desejado.

Uma vez que o Angular cria aplicações SPA (Single Page Applications), o roteamento permite navegar entre componentes e até simular que as páginas estão a ser recarregadas mesmo quando não o estão a ser.

2.4.1.8 Testes

Esta *framework* também tem ferramentas de testes incorporadas que permitem realizar certas tarefas programadas para verificar se está tudo a correr corretamente. Permite realizar testes de serviços e *pipes* isoladas, assim como de módulos completos.

2.4.2 ReactJS

O React, ao contrário do Angular, não é uma *fremework* completa, mas sim uma biblioteca de front-end para construir interfaces.

Esta é declarativa (permite declarar *views* que tornam o código fácil de fazer *debug* e mais previsível) e é também baseada em componentes (semelhante ao Angular)

Esta biblioteca é baseada em Javascript e tem, na sua constituição, várias funcionalidades que serão descritas seguidamente.

2.4.2.1 JSX

O JSX é uma extensão de sintaxe do Javascript que permite declarar variáveis do género representado na Figura 2.14.

```
const element = <h1>Hello, world!</h1>;
```

Figura 2.14 - Sintaxe em JSX (Facebook, n.d.-a)

Isto permite juntar componentes do template HTML com componentes de Javascript e realizar logo todas as tarefas numa só declaração.

2.4.2.2 Renderização de Elementos

Uma vez que os elementos do React são bastante simples, é necessário declarações adicionais para permitir que estes sejas injetados no DOM.

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Figura 2.15 - Declaração de renderização de elemento (Facebook, n.d.-b)

Com esta declaração, o elemento vai ser colocado no DOM para que o utilizador o possa visualizar, gerando "Hello World".

2.4.2.3 Componentes

O React também possui a divisão de código por elementos.

Criando esta divisão, é possível criar blocos de código independentes para que estes não estejam de maneira nenhuma dependentes de outros elementos, tornando o código mais simples de testar e compreender.

2.4.2.4 Formulários

Os formulários também funcionam de uma forma diferente com o React.

Estes formulários dinâmicos permitem que seja guardado um estado comum ao formulário e não apenas um estado comum a cada elemento como "<input>" ou "<select>". Estes estados apenas podem ser alterados com o setState().

2.4.3 Vue.js

O Vue é uma *framework* de desenvolvimento *web* progressiva para construir interfaces (You, n.d.).

Esta ferramenta é possível de ser integrada com outras bibliotecas assim como tem a capacidade de construir SPA's se for combinada com outras ferramentas e bibliotecas.

O Vue é a terceira força de desenvolvimento web crescente, atrás do React e do Angular 2+ e está cada vez mais a ser adotada por diferentes programadores.

À semelhança do Angular e do React, o Vue.js também tem alguns princípios fundamentais para a construção de aplicações.

2.4.3.1 Componentes, Formulários, Renderização e Gestão de eventos

Os componentes são a ferramenta mais poderosa do Vue, uma vez que eles permitem o encapsulamento de código reutilizável em outros componentes e adiciona comportamentos definidos ao HTML básico.

Através dos formulários do Vue é possível criar formulários cuja troca de dados é realizada nos dois sentidos entre o DOM e o Vue. O último deteta automaticamente a forma correta de atualizar os elementos do formulário baseando-se no tipo de *input*.

A renderização condicional é, basicamente, a introdução de "if's" em desenvolvimento web. Através da verificação de variáveis pode-se determinar se um elemento é ou não mostrado no DOM.

O gestor de eventos permite que determinado evento do DOM esteja a ser monitorizado e, quando este é detetado, é realizado algum código em Javascript em reação ao evento.

2.4.4 Conclusões e comparações

Para as *frameworks* de desenvolvimento, foram escolhidos alguns fatores decisivos na escolha da ferramenta a utilizar de acordo com o projeto.

Para grandes projetos nota-se que o Angular e o React são as escolhas óbvias, porque permitem suporte a grandes equipas, mas para programadores individuais com pouca experiência em Javascript, o Vue é a ferramenta mais acessível.

O Angular é o escolhido nesta dissertação porque é a *framework* mais completa e o typescript é o que mais se aproxima com a maioria das linguagens de programação mais focadas no Mestrado Integrado em Engenharia Eletrotécnica e de Computadores.

Tabela 2.2 - Características das frameworks a utilizar no protótipo

Software	Curva de Aprendizagem	Tamanho da equipa a desenvolver	Modular	SPA (Single-Page Application)	Linguagem
Angular 2+	Média/Alta	grande	Sim	Sim	Typescript (Permite o uso de Javascript ES6)
ReactJS	Rápida/Média	média/grande	Sim	Sim	Javascript ES6
Vue.js	Rápida	pequena	Sim	Sim	Javascript ES5, ES6 e Typescript

3

3 Proposta conceptual

3.1 Identificação do problema

A falta de uma ferramenta centralizadora da informação é um dos problemas que muitas empresas sofrem. Problema este existente devido a falta de fundos ou simplesmente devido à falta de pessoal qualificado que possa planear e desenvolver uma solução deste género. Muitas vezes nota-se também que o principal problema é, simplesmente, falta de planeamento que, a longo prazo, pode causar problemas estruturais na organização dos trabalhadores.

A comunicação entre os trabalhadores deverá ser realizada da forma mais simples e rápida possível, permitindo que estes não percam mais tempo nesta comunicação do que efetivamente no seu trabalho.

Este é um aspeto fundamental que muitas ferramentas no mercado simplesmente não respeitam. Estas necessitam, quase sempre, de um curso antes da utilização do software, cursos estes que também, na sua generalidade, não são muito apelativos e não conseguem abranger a capacidade de absorção de conhecimento dos diferentes tipos de pessoas.

Para tentar responder a este problema, será criada uma aplicação com rotinas de trabalho simples para tarefas que, embora pequenas, podem tornar muito mais eficientes as tarefas dos trabalhadores e o seu dia de trabalho.

O objetivo não é criar um super software com todas as funcionalidades que uma empresa necessita mas sim criar um software, nesta fase, focado à gestão de processos internos das empresas.

3.2 Requisitos da aplicação

Os requisitos a definir para a solução do problema vão-se dividir em **requisitos funcionais** e **não funcionais**.

Como definido em (Rainardi, 2007), requisitos funcionais são aqueles que definem o que a aplicação vai fazer, como problemas a que a última vai responder e a resposta que esta vai dar dada a interação com os utilizadores.

Os requisitos não funcionais são os requisitos que não caracterizam diretamente funções/ações da plataforma, mas são características que esta deverá ter para funcionar como pretendido.

3.2.1 Requisitos funcionais

Os requisitos funcionais definidos nesta aplicação devem ter como base a simplicidade do cumprimento das tarefas a serem definidas.

Como tal, define-se os seguintes requisitos:

- A aplicação deverá ser desenvolvida em plataforma web para evitar instalações locais nas máquinas dos utilizadores;
- Deverá haver a possibilidade de criar utilizadores com diferentes tipos de forma a poder discernir acessos a funcionalidades entre os mesmos;
- O sistema deverá guardar dados resultantes da interação com o utilizador numa localização externa;
- A plataforma deverá ser modular e não dependente entre módulos de forma a conseguir ativar apenas as funcionalidades que os utilizadores desejam;
  - A plataforma de *back-end* deverá ser o mais genérica possível para conseguir generalizar a troca de dados com a mesma e, possivelmente, aplicar a outras plataformas.

3.2.2 Requisitos não funcionais

Os requisitos não funcionais definidos têm como objetivo a simplificação da plataforma, interação com os utilizadores e possibilidade de acrescento de módulos no futuro.

- A interface da aplicação deverá será a mais simples e intuitiva possível de forma a que o utilizador consiga perceber para onde tem de ir com pouca ou nenhuma explicação;
- A ferramenta deverá garantir que a liberdade de inserção de dados não dificulta a compreensão da interação dos utilizadores.
- A estruturação do código deverá ser explícita para permitir a compreensão de outros programadores;
- A aplicação deverá permitir o desenvolvimento de novos módulos sem que seja necessário alterar algo nos já existentes.

3.3 Solução proposta

A solução proposta passa então por uma aplicação web para ser o mais abrangente possível e poder ser utilizada em múltiplos dispositivos sem necessidade de instalações locais.

A interface da mesma será o mais simples possível, isto é, a interação com o utilizador será fácil e intuitiva, mesmo que as rotinas por detrás disso não o sejam.

A aplicação será modular e permitirá acrescentar novos módulos no futuro consoante a necessidade da empresa.

Os utilizadores serão criados com determinados tipos, tipos estes que darão acesso a certas funcionalidades definidas pelos administradores dos projetos.

3.3.1 Estrutura da Aplicação

A aplicação a desenhar terá a estrutura como base da Figura 3.1.

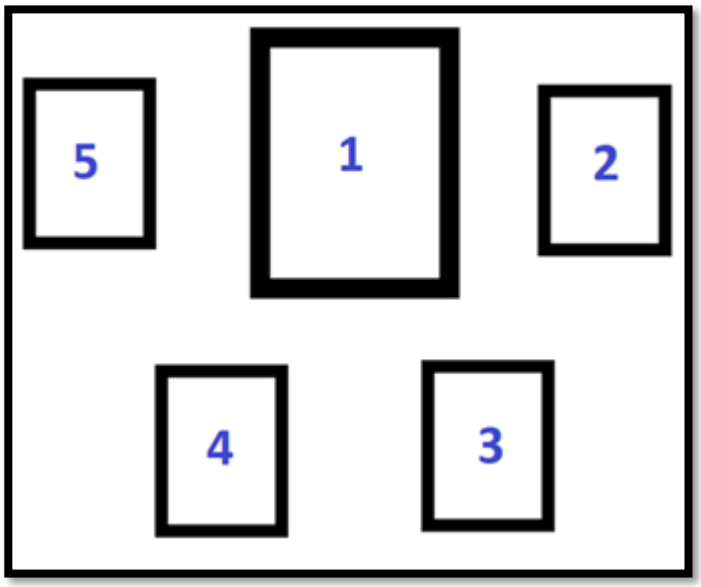

Figura 3.1 - Desenho básico da aplicação

Como se pode observar na Figura 3.1, a aplicação está dividida em módulos. Todos estes módulos são independentes entre sim e qualquer um deles poderá ser removido sem causar qualquer transtorno à aplicação.

Numa outra visão da aplicação, pode-se observar a Figura 3.2.

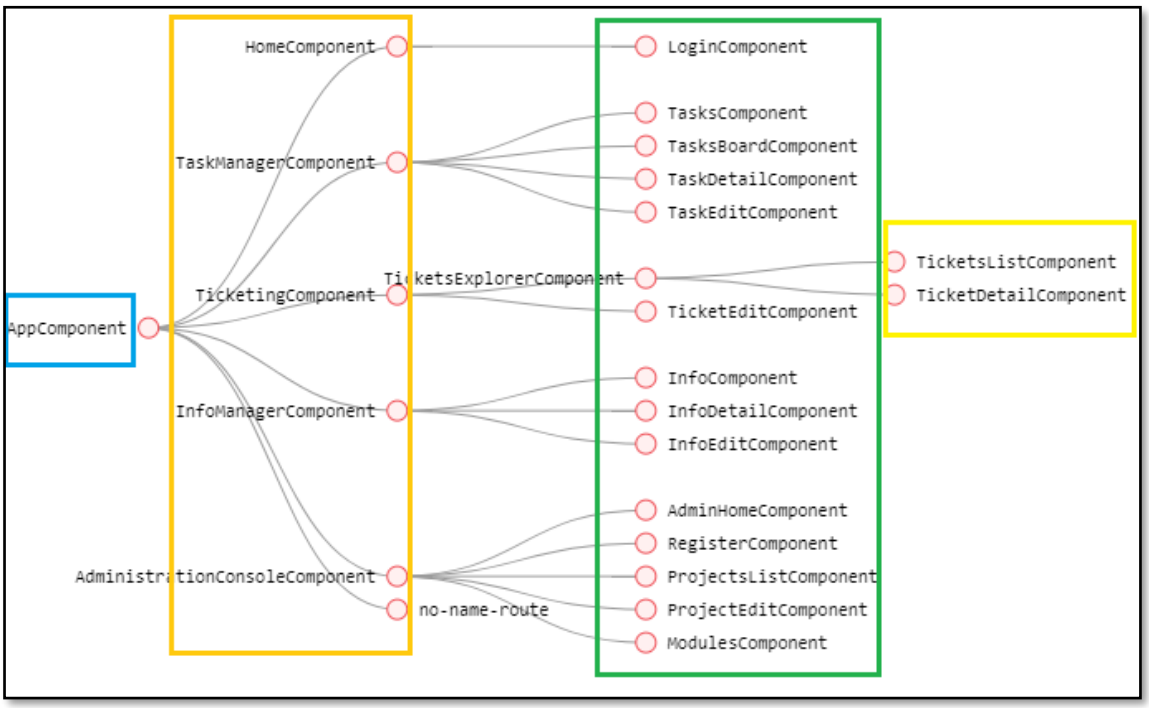

Figura 3.2 - Estrutura em árvore da aplicação

Nota-se então que a aplicação está distribuída em 4 níveis principais.

O nível azul corresponde apenas ao componente, em Angular (será especificado em maior detalhe posteriormente), que introduz todos os outros componentes.

A laranja, estão os componentes principais da aplicação e um componente "no-name-route" que corresponde a todos os caminhos que a aplicação não tiver roteados, redirecionando para o HomeComponent.

A verde e a amarelo estão os componentes característicos de cada componente principal, isto é, as funcionalidades de cada módulo.

A estrutura em árvore definida na Figura 3.2 tem como base as funcionalidades e a estrutura organizacional da *framework* Angular 2+. Foi optado este caminho uma vez que permite uma fácil visualização da estrutura da aplicação, tanto para quem tem conhecimento sobre ela como para quem não tem.

Passa-se então, seguidamente, para explicação detalhada de cada um dos módulos.

3.3.1.1 Centro da Aplicação

O centro da aplicação (HomeComponent) é a área inicial a que o utilizador tem acesso cada vez que efetua o login na aplicação. Este componente redireciona para todos os outros (exceto a consola de administração) e permite guiar o utilizador para uma correta utilização das funcionalidades, redirecionado o utilizador para a rota pretendida.

Este módulo tem duas fases aplicadas (Figura 3.3), uma quando o utilizador não tem o login efetuado e outra quando o login é efetuado.

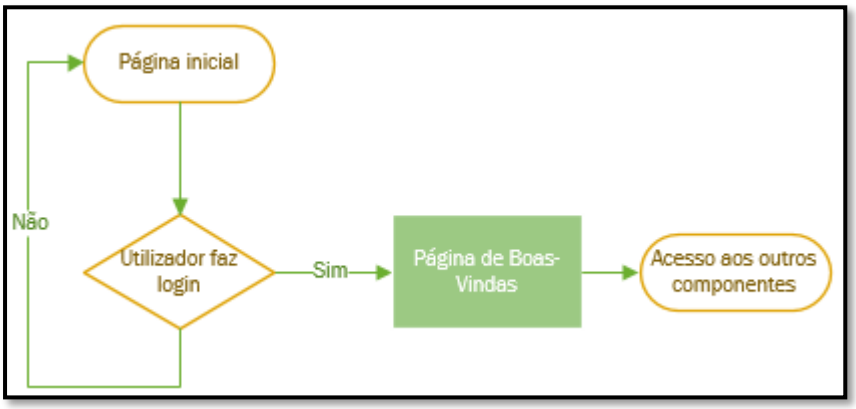

Figura 3.3 - Fluxograma funcional do centro da aplicação

Este módulo, sendo o mais básico, também apresenta o fluxograma funcional mais básico com poucos passos que o utilizador poderá dar.

3.3.1.2 Gestor de tarefas

No gestor de tarefas é possível criar e gerir tarefas de acordo com a secção/departamento do utilizador e conforme as permissões que lhe foram atribuídas.

Este módulo está dividido em dois modos de visualização de dados. Um modo de listagem e um modo em Kanban-Board (modo este que permite uma visualização mais esquematizada das tarefas abertas em cada projeto.

Cada projeto criado na aplicação terá a sua secção de tarefas, sendo que cada utilizador, conforme a sua área, pode estar inserido em um ou múltiplos projetos conforme o que for atribuído na criação/edição do projeto.

As três fases principais deste módulo são definidas por:

- **Escolha do projeto –** Quando o utilizador entre nesta módulo é apresentada a opção para escolher o projeto que o utilizador deseja ver. Apenas são apresentados os projetos a que o utilizador tem acesso;
- **Visualização em lista –** Posta a escolha do projeto, o utilizador pode escolher a visualização em lista das tarefas;
- **Visualização em Kanban-Board –** O utilizador também poderá escolher este modo de visualização de tarefas.

Para além destas três fases principais, ainda é possível:

- Criar tarefas;
- Editar tarefas;
- Apagar tarefas;
- Visualizar cada tarefas num componente independente;
- Procurar um tarefas por ID;
- No modo de listagem, organizar as tarefas por estado, isto é, filtrar as tarefas que deseja ver conforme o seu estado (aberto, em progresso, etc…).

Ao trocar de módulo, o projeto que o utilizador tinha escolhido não é perdido e ao voltar a entrar no gestor de tarefas, é apresenta o último projeto escolhido.

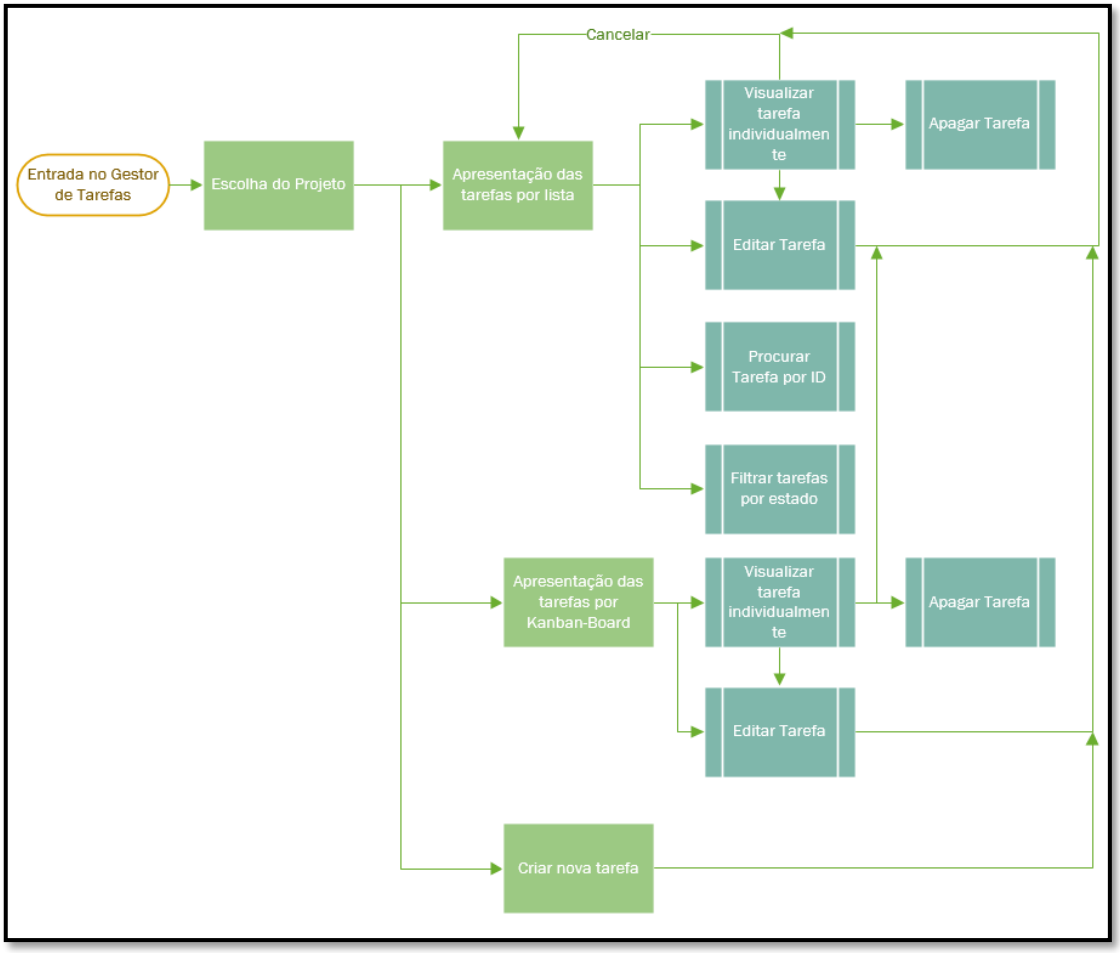

Figura 3.4 - Fluxograma funcional do gestor de tarefas

Na Figura 3.4 pode-se ver o esquemático simplificado do funcionamento do gestor de tarefas. Este está definido, maioritariamente, por três linhas de ação de forma a simplificar a vida ao utilizador e a facilitar a compreensão da ferramenta.

3.3.1.3 Ticketing

A constituição deste módulo é semelhante à do gestor de tarefas mas cada ticket terá uma descrição de campos mais detalhada e estes tickets apenas poderão ser abertos para outros projetos.

A projeção da informação para o utilizador, neste módulo, passará por três fontes diferentes de visualização:

- **Visualização de todos os tickets acessíveis –** Neste modo é possível visualizar todos os tickets a que o utilizador tem acesso;
- **Visualização por projeto –** Aqui o utilizador poderá visualizar os tickets correspondentes a determinado projeto a que ele tem acesso;
- **Visualização geral de qualquer projeto –** O que distingue este modo do anterior é a capacidade de ver os tickets abertos correspondentes a qualquer projeto e não só os tickets a que o utilizador tem acesso. Optou-se por este modo de visualização uma vez que a confidencialidade de tarefas abertas a outros departamentos, no geral, não é elevada e, como tal, qualquer utilizador poderá ver os tickets abertos em qualquer situação.

Relativamente aos projetos de um utilizador, este ainda terá mais opções de visualização. Ao escolher um projeto ou estado na visualização de todos os tickets, poder-se-á optar por visualizar os tickets que foram abertos através de projetos externos ou os tickets que foram abertos pelo projeto em questão.

À semelhança da listagem de tarefas, também é possível filtrar os tickets por ID e por estado, criar tickets e visualizar os tickets individualmente.

A abertura de um ticket só poderá se realizada de um projeto para outro projeto e todos os utilizadores com acesso a cada um deles poderá editar os tickets criado e manter-se a par das atualizações dos tickets.

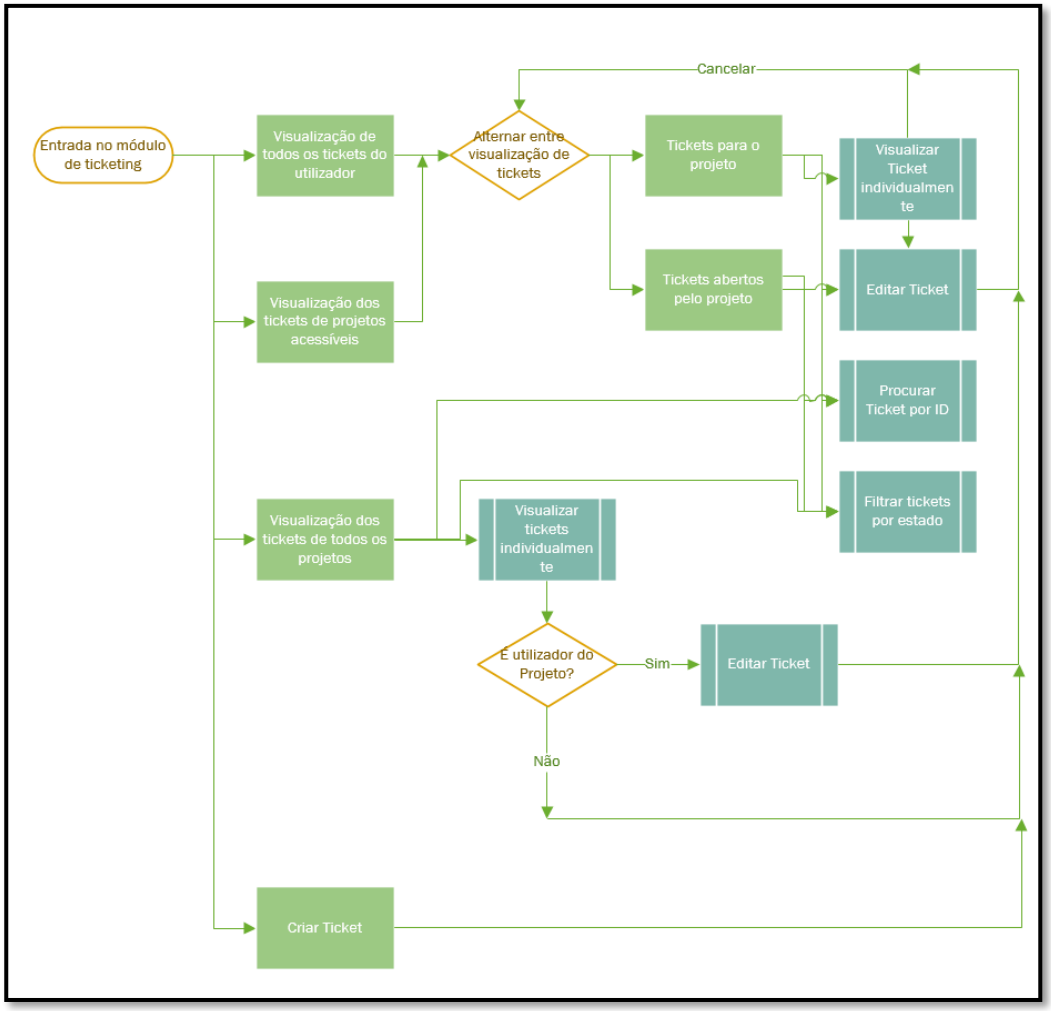

Figura 3.5 - Fluxograma funcional de ticketing

O fluxograma representado na Figura 3.5 retrata, de forma simplificada, o funcionamento da ferramenta de Ticketing. Aqui já se denota uma maior complexidade das decisões que o utilizador pode tomar e não é extremamente trivial a navegação neste painel. No entanto, a interface é intuitiva e mesmo sem explicação o utilizador consegue encontrar o caminho que deseja seguir.

3.3.1.4 Gestor de informação

Este componente tem como principal função o armazenamento e consulta de qualquer tipo de informação que a empresa/projeto necessite. Aproveita-se a lógica de centralização de informação para adicionar este módulo versátil à solução.

Dependendo da necessidade, vários níveis de utilizadores poderão inserir informação nesta plataforma e a sua consulta, também dependendo da sensibilidade da informação, poderá também ser limitada a certa área/utilizadores.

Só está disponível a cada utilizador o projeto a que tem acesso e, tendo acesso ao projeto, é possível inserir, editar e apagar informação, sempre com a mesma estrutura de dados e ficando ao critério dos utilizadores o tipo de informação que colocam lá.

A divisão deste módulo é bastante simples:

- **Escolha do projeto;**
- **Escolha da secção de informação –** Cada secção tem informações individuais, isto é, a informação em cada secção é independente e, trocando de secção, troca-se a informação.

A versatilidade deste módulo está na adição e remoção de secções. Assim, é da responsabilidade do utilizador colocar a informação que lhe dará mais benefícios não dependente de um *template* padrão.

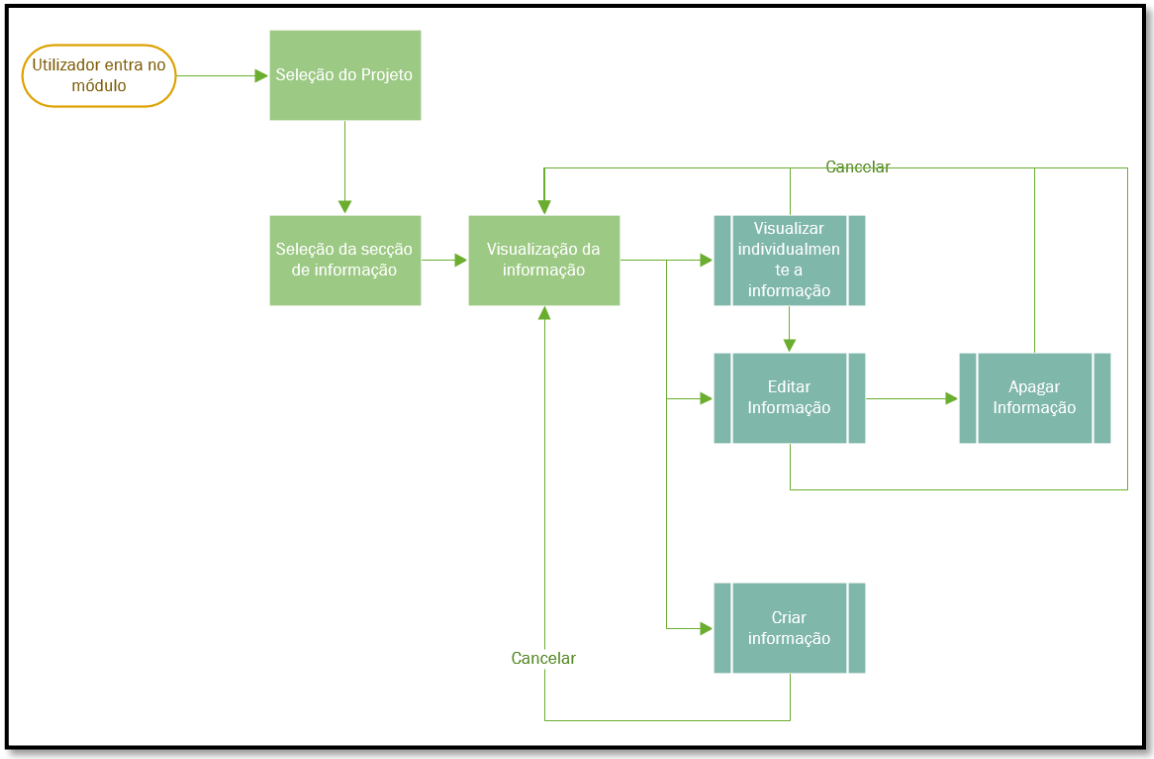

Figura 3.6 - Fluxograma funcional do gestor de informação

Sendo este o segundo módulo mais simples, o a sua simplicidade pode ser verificada no fluxograma, tendo apenas dois níveis importantes de decisão do utilizador. Aqui primou-se pela simplicidade porque o objetivo é que o utilizador chegue rápido à informação que deseja ver.

3.3.1.5 Consola de Administração (5)

A consola de administração é o componente onde o/os administradores da aplicação/projetos terão o controlo total sobre a mesma.

Este é o módulo de ligação de baixo nível entre todos os outros, onde as tarefas administrativas poderão ser possíveis de se realizar.

As funcionalidades disponíveis neste módulo passam por:

- Registar utilizadores (apenas administradores);
- Alterar parâmetros do utilizador;
- Criar projetos (apenas administradores);
- Editar projetos (administradores da aplicação e do projeto);
- Ativar e desativar módulos.

A consola de administração é composta por muitas pequenas tarefas e, como tal, o utilizador não tem um caminho predefinido para chegar ao objetivo final, tem vários. Quando o utilizador entra no módulo, a interface propõe-lhe que escolha uma opção para dar início ao trabalho.

O registo de utilizadores, como foi referido, só poderá ser feito pelos administradores da aplicação e a alteração de um utilizador só poderá ser feito pelo próprio utilizador. A alteração da password poderá ser realizada pelos administrador mas apenas na consola de administração do painel de *back-end*.

A atualização do utilizador tem várias restrições que não permitem que o mesmo troque o utilizador para um já existente, assim como um email já existente. Estas regras também se aplicam para a criação de utilizadores.

A criação de projeto é um processo simples que vai criar o elo de ligação dos restantes módulos. Aqui é necessário escolher a área da empresa que terá acesso ao projeto e os administradores do projeto. Apenas o campo do nome, descrição e a área de acesso são campos obrigatórios.

Para editar um projeto, será apresentada uma lista dos projetos a que o utilizador tem acesso e ele poderá optar por editar ou apagar o projeto.

Por fim e para tornar a utilização da aplicação mais centrada à necessidade da empresa, os administradores da aplicação podem desativar o acesso aos módulos na consola de administração.

3.3.1.6 Autenticação

A autenticação é uma das funcionalidades mais importantes nesta ferramenta.

A autenticação do utilizador é que vai permitir aceder aos dados guardados na base de dados e assim conseguir navegar na aplicação.

A implementação deste módulo será falada em maior detalhe no capítulo 4 e a direção que o utilizador poderá tomar conforme a autenticação é bastante simples:

- **Utilizador autentica-se –** Tem acesso a todos os módulos ativos da aplicação e pode navegar livremente;
- **Utilizador não se autentica –** Não tem qualquer tipo de permissão na aplicação.

Como é referido, devido a esta restrições, a autenticação está na base de toda a utilização da ferramenta. Em baixo nível é o que permite retirar os dados da base de dados e mais a alto nível é o que restringe determinados acessos aos utilizadores.

Pode-se então dividir o tipo de autenticação presente em duas categorias:

- **Autenticação de baixo nível –** Implementação incorporada no *back-end* a que o *front-end* acede e troca informação;~

  • **Autenticação de alto nível –** Implementação efetuada no front-end de forma a conseguir restringir acessos aos utilizadores. A razão para a existência destes dois tipos de autenticação será referida também no capítulo 4 da implementação.

3.3.2 Cenários de utilização

3.3.2.1 Pequena empresa (+/- 15 trabalhadores)

Necessidade

A empresa necessita de uma pequena ferramenta para a organização de tarefas entre os colaboradores, assim como uma ferramenta que permite guardar, internamente, alguma informação relativa a recursos disponíveis.

Caso de Uso

Uma vez que a aplicação em estudo é modular, nesta situação desativar-se-ia os módulos de **Ticketing** e de **Gestão de Pessoal e Custos**.

A empresa em questão usaria apenas o módulo do **Gestor de Tarefas** e o **Gestor de Informação**.

No primeiro, caso fosse necessário, criar-se-ia variadas secções, consoante os departamentos envolvidos na empresa, e cada secção teria acesso às tarefas introduzidas no gestor de tarefas.

No segundo, também se necessário, seriam criadas essas secções para que cada secção correspondente tivesse acesso à informação que lhe seria respetiva. Assim, conforme a autenticação, a informação a consultar estaria de acordo com o tipo de utilizador.

3.3.2.2 Média empresa (+/- 100 trabalhadores)

Necessidade

Esta empresa está com dificuldades em gerir a troca de informação entre departamentos para manter um fluxo de trabalho dinâmico e eficiente.

Caso de Uso

Para resolver estes dois problemas utilizar-se-ia o módulo de **Ticketing.**

Este módulo ajudaria a resolver o problema de comunicação. Sempre que fosse necessário resolver algum problema num departamento e a resolução deste problema fosse destina a outro, um pedido seria aberto neste componente da

aplicação e ficaria associado ao departamento destinatário. Assim, ficaria registado o problema, data e destinatário, assim como a prioridade, e restaria apenas à pessoa encarregue consultar o pedido e resolvê-lo.

3.3.2.3 Grande Empresa (+ de 1000 trabalhadores)

Necessidade

Nesta empresa sentem-se as necessidades das duas anteriores, uma vez que a sua dimensão é muito maior e pretende-se resolver os problemas usando a aplicação desenvolvida.

Caso de Uso

Nesta situação, todos os módulos seriam utilizados.

O **Gestor de Tarefas** seria usado em toda a empresa e nos diferentes departamentos, sendo que cada um só teria acesso às tarefas que lhe são correspondentes. Aqui as tarefas seriam usadas para gerir o trabalho de cada colaborador e ajudá-los-ia a não se esquecer de algumas tarefas enquanto desenvolviam outras.

O módulo de **Ticketing** seria usado de uma maneira muito semelhante ao anterior. Os departamentos poderiam reportar problemas/tarefas uns aos outros e caberia à pessoa destinatária fechar o pedido.

Por fim, o módulo de **Gestão de Informação** manteria atualizadas as aplicações que cada departamento usa, assim como, se necessário, informações relativas a infraestruturas utilizadas pelas a empresa, como IP's de servidores, tipo de servidor, etc.

4

4 Implementação

Neste capítulo pretende-se detalhar a metodologia de desenvolvimento da solução proposta, assim como serão apresentadas as ferramentas de desenvolvimento, estrutura de dados e ferramenta de *back-end* e os algoritmos utilizados para implementar as funcionalidades da aplicação.

4.1 Ferramentas de desenvolvimento

Para criar uma aplicação baseada na web, são sempre necessários dois níveis de desenvolvimento: *front-end* e *back-end*.

A camada de *front-end* é o nível visual, o nível que passa diretamente para o utilizador e trata dos dados gerados. A camada do *back-end* é a camada inferior, normalmente onde se guarda dados, onde se gere autenticações e outras regras necessárias ao funcionamento da aplicação.

Uma vez que a componente principal desta solução passa pela interação com o utilizador, o foco principal é na ferramenta de *front-end*, **Angular**. Como é necessário a segunda, optou-se por utilizar como *back-end* a solução criada pela Google: **Firebase**.

A razão principal pela escolha do Angular como ferramenta principal é o facto desta ser uma *framework* completa para o efeito, em comparação com as outras opções estudadas. Como no Angular já estão disponíveis muitas funcionalidades sem ter de recorrer a bibliotecas externas, torna mais fácil a implementação

da aplicação. O facto de, das três estudas, ser a única que utiliza Typescript também é uma vantagem. Este tipo de linguagem é muito semelhante ao Javascript mas permite a declaração de tipos e é compilado antes de correr ao vivo. Isto permite fazer um debugging mais rápido e simples de perceber, ao contrário das outras ferramentas puramente baseadas em Javascript.

O seu antecessor, AngularJS é ainda uma das *frameworks* mais utilizadas no mercado para a criação de soluções web, dando isto uma maior força à utilização futura do Angular e reforçando a escolha do mesmo.

O facto de todas as novas *frameworks* de web permitirem uma construção de código modular também dá força à escolha de qualquer uma delas porque um dos objetivos desta solução é ter capacidade, no futuro, para permitir novos updates sem que seja preciso alterar muita coisa no código já anteriormente feito.

Relativamente ao Firebase também foi uma escolha óbvia. Era necessária uma ferramenta de *back-end* que já tivesse rotinas pré-definidas, uma interface fácil e intuitiva de perceber e que houvesse a possibilidade de integrar com o Angular. É aqui então que entra o Firebase. Este já tem uma interface fácil de utilizar, rotinas de autenticação de utilizadores pré-definidos e, acima de tudo, tem uma base de dados dinâmica que permite armazenar a informação gerada na aplicação.

Seguidamente irão ser detalhadas as funcionalidades utilizadas no Angular e no Firebase para esta solução.

4.1.1 Angular

O Angular dispõe de funcionalidades extremamente importantes que facilitam e reinventam o desenvolvimento web. Não utilizando todas estas que estão disponíveis, foi possível utilizar uma grande maioria delas, estando as mesmas representadas na Tabela 4.1.

Tabela 4.1 - Tabela de funcionalidades utilizadas e as suas aplicações

Funcionalidades	Aplicações
Routing	Permite navegar dentro da aplicação, alternando entre os caminhos no navegador.
Serviços	Fazem a troca de informação entre módulos. Esta funcionalidade é a ponte de ligação entre os componentes.
Formulários Reativos	São uma variante dos formulários normais presentes na *framework* mas permitem um maior controlo sobre a informação gerada nos formulários da aplicação.
Client Http	Funcionalidade que permite fazer a troca de informação com a ferramenta de *back-end*.
Modelos	Os modelos são semelhantes a estruturas de dados que são utilizados para definir objetos mais complexos.
Tokens	String que permite fazer a validação da autenticação dos utilizadores.
Intercetores	Permite fazer a interceção de pedidos de informação ao *back-end* e utiliza os tokens para validar se o utilizador tem autorização para aceder à plataforma.
Subjects	É uma funcionalidade que permite fazer a comunicação em tempo-real entre componentes.
Guards	São rotinas criadas que permitem bloquear ou autorizar o acesso a determinado caminho da aplicação.
Two-way Databinding	Utilizado para fazer a troca de dados entre o Angular e o DOM

Diretivas	São diretivas que permitem alterar a aparência ou comportamento de um elemento do DOM.
Injeção de Módulos e Serviços	Permite chamar módulos e serviços noutros serviços

No decorrer do capítulo serão detalhados os casos em que se utilizam as funcionalidades indicadas na tabela e como é que elas funcionam.

4.1.2 Firebase

O Firebase foi utilizado com duas funções principais:

- Autenticar utilizadores;
- Guardar dados de utilização.

A autenticação dos utilizadores foi feita via o método de email/password e os dados foram guardados numa base de dados não-SQL fornecida pelo Firebase.

Para acesso à base de dados, utilizou-se as duas regras seguintes de acesso:

```
"rules": {
  ".read": "auth != null",
  ".write": "auth != null'
}
```

Figura 4.1 - Regras de acesso à BD

A regra da Figura 4.1 implica que só os utilizadores autenticados é que podem ler ou escrever na base de dados.

4.2 Estrutura da Aplicação

Um dos critérios definidos para a implementação da solução é a modularidade da mesma. O primeiro passo para estruturar a aplicação foi então criar componentes independentes, cada um com a sua função definida.

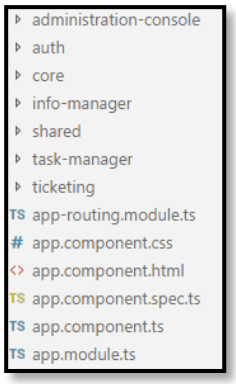

Figura 4.2 - Estrutura modular da aplicação

A Figura 4.2 mostra a organização dos componentes dentro da estrutura de código do Angular. O componente "administration-console" tem todo o código referente à consola de administração. No componente "auth" é implementado todo o código de autenticação. No "core" estão simplesmente presentes a página principal "HomeComponent" e o header sempre presente na aplicação. O "info-manager", "task-manager" e "ticketing" são, respetivamente, o Gestor de informação, Gestor de Tarefas e Ticketing e, por fim, a pasta "shared" tem código e funcionalidades relevantes a toda a aplicação que é utilizado em diferentes sítios. Ainda nesta figura, temos representado o "app-component" que é o componente raíz de toda a aplicação.

Na Figura 3.2 estão representadas todas as ligações entre componentes e uma estrutura mais detalhada de como o código está organizado na aplicação.

4.3 Estrutura e Interação de Dados

A estrutura de dados é o primeiro passo a definir na implementação da aplicação. Como tal, existem diferentes objetos que se pretende guardar na base de dados, sendo estes:

- **User**:
    - username (string);

    - role (string);
    - email (string);
    - password (string).
- **Project**:
    - name (string);
    - description (string);
    - roleAccess (string[]);
    - infoSections (string[]);
    - administrators (string[]).
- **Task:**
    - id (number);
    - name (string);
    - details (string);
    - priority (string);
    - reporter (string);
    - assignee (string);
    - state (string);
    - project (Project).
- **Ticket:**
    - id (number);
    - summary (string);
    - description (string);
    - ticketReporter (Project);
    - destProject (Project);
    - priority (string);
    - state (string).
- **Info:**
    - summary (string);
    - data (string);
    - observations (string);
    - project (string);
    - section (string).
- **Module:**
    - name (string);

- status (string).

A Figura 4.3 mostra os modelos da aplicação que são guardados na base de dados quando as interações assim o necessitam.

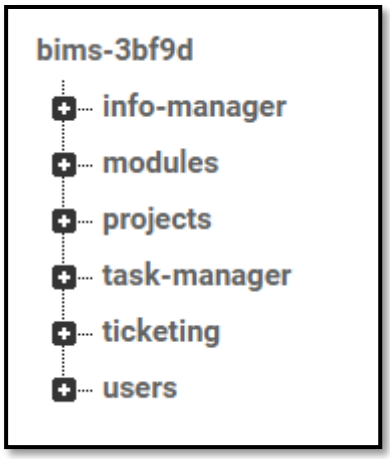

Figura 4.3 - Estrutura de dados na base de dados

Para conseguir guardar os dados é necessário ligar a API do Firebase ao Angular ao iniciar a aplicação (Figura 4.4).

```
ngOnInit() {
  firebase.initializeApp({
    apiKey: 'AIzaSyABrMjW8wNCzNAyHz8JOh2zVjtcOorhEgI',
    authDomain: 'https://bims-3bf9d.firebaseio.com/',
  });
```

Figura 4.4 - Dados da API gerados automaticamente no Firebase

Posta a autenticação e todas as verificações necessárias, utiliza-se o serviço **"DataStorageService" (**Figura 4.5) para efetuar toda a troca de dados com o *back-end*. Este serviço tem três métodos principais de troca de informação, sendo que estes se repetem trocando apenas variáveis, sendo estes métodos:

Store

```
storeUsers () {
  const req = new HttpRequest('PUT', 'https://bims-3bf9d.firebaseio.com/users.json', this.authService.getUsers(), {
    reportProgress: true,
  });
  return this.httpClient.request(req);
}
```

Figura 4.5 - Método para guardar utilizadores na BD

Este método faz a ligação via Http ao Firebase e efetua um **PUT** de todos os utilizadores na base de dados. O objeto que guarda estes utilizadores está sempre no front-end e, quando existem alterações nos utilizadores, é chamado este método que envia o objeto para o *back-end*.

Get

```
getUsers(email: string) {
  this.httpClient.get<User[]>('https://bims-3bf9d.firebaseio.com/users.json', {
    responseType: 'json'
  })
  .map(
    (users) => {
      return users;
    }
  )
  .subscribe(
    (users: User[]) => {
      this.authService.setUsers(users);
      this.authService.setLoggedUser(this.authService.getUserbyEmail(email));
      this.authService.usersLoaded.next(true);
    },
    error => {
      this.authService.token = null;
      this.authService.usersLoaded.next(false);
    }
  );
}
```

Figura 4.6 - Método para receber utilizadores da BD

Neste método (Figura 4.6), ao contrário do outro, recebe-se a informação da base de dados e guarda-se no local correspondente da aplicação que, neste caso, corresponde aos utilizadores.

Remove

```
removeProject (index: number) {
  this.projectsService.deleteProject(index);
  this.httpClient.delete('https://bims-3bf9d.firebaseio.com/projects/' + index + '.json')
  .subscribe(
    (val) => {
        console.log('The project was deleted');
    });
}
```

Figura 4.7 - Método para remover um projeto na BD

A Figura 4.7 mostra o último método utilizado quando é necessário remover algo da base de dados.

4.4 Autenticação

A autenticação dos utilizadores é o processo mais importante da aplicação, uma vez que salvaguarda o uso para quem deve ter acesso e garante, num segundo nível, o acesso a certas funcionalidades a utilizadores já autenticados.

Após o utilizador realizar o login, os utilizadores são carregados para o serviço "**AuthService"** e permite que todos os outros componentes possam ter acesso ao objeto dos utilizadores.

Para atingir o efeito pretendido, a autenticação divide-se nos vários níveis descritos seguidamente.

4.4.1 Login

O login de um utilizador (Figura 4.8) é efetuado em dois níveis.

Nível 1

Através do *back-end* é utilizado o método de autenticação "Email/Password", isto é, o utilizador coloca o email e password para se autenticar e o *back-end* confirma se os dados estão corretos, gerando um **token de autenticação** após o sucesso.

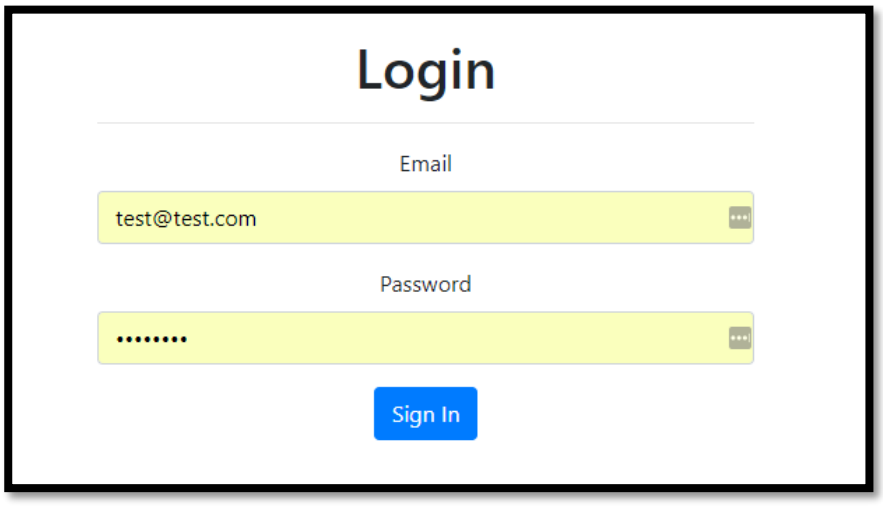

Figura 4.8 - Componente inicial de Login

Nível 2

Neste nível, após o sucesso do login, é efetuado carregamento de todos os utilizadores da aplicação, recorrendo à base de dados. A estrutura de dados de cada utilizador, definido em 4.3, vai ser utilizada neste segundo nível para validar se o utilizador tem acesso a determinadas funcionalidades recorrendo ao **role** e **username** do utilizador.

4.4.2 Registo de utilizadores

O registo de um utilizador (Figura 4.10) é feito na consola de administração, onde apenas os administradores da aplicação o podem fazer. O registo é então dividido em duas partes: **validação pré-registo** e **comunicação do registo ao *back-end*.**

Na validação do registo ()Figura 4.9, utiliza-se um algoritmo que valida se o username ou o email já existem. Existindo, falha o registo, não existindo, o processo segue para a segunda parte.

```
// registering a new user
if (!this.authService.updatingUser) {
  if (this.authService.userExists(user.username)) {
    if (this.authService.emailExists(user.email)) {
      this.authService.addUsers(user);
      this.storeUsersSub();
      this.authService.signupUser(this.registerForm.value.email, this.registerForm.value.password);
      return true;
    }
    console.log('Email already exists');
  }
  console.log('Username already exists');
}
```

Figura 4.9 - Algoritmo de validação de registo de utilizador

Na comunicação do registo, a aplicação recorre ap **"AuthService"** (serviço de autenticação), onde aplica o método de comunicação com o Firebase e redireciona o utilizador para o centro da aplicação.s

```
signupUser(email: string, password: string) {
  firebase.auth().createUserWithEmailAndPassword(email, password)
  .then(
    response => {
      this.router.navigate(['/']); // navigate away
    }
  )
  .catch(
    error => console.log(error) // catches error
  );
}
```

Figura 4.10 - Criação de utilizador no Firebase

4.4.3 Atualização de utilizadores

Esta funcionalidade (Figura 4.11) pode ser feita por qualquer utilizador, alterando apenas os campos que lhe correspondem e também são feitas verificações semelhantes ao do registo, no entanto tem que se cruzar as variadas hipóteses (ex: existe utilizador e não existe email, existe email e não existe utilizador, etc…) tornando o algoritmo mais extenso e falhando a atualização caso algum dos casos retorne verdade.

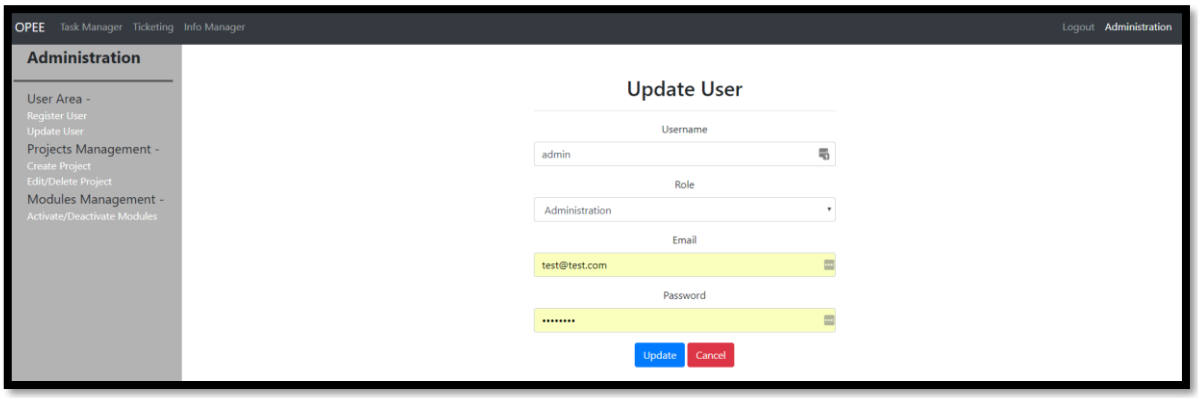

Figura 4.11 - Módulo de atualização do utilizador

4.4.4 Logout

Para um utilizador fazer logout é necessário apenas chamar um pequeno método (Figura 4.12) que torna inválida a autenticação corrente.

```
logout() {
  firebase.auth().signOut();
  this.token = null;
  this.loggedUser = null;
  this.usersLoaded.next(false);
}
```

Figura 4.12 - Método de logout

O **token** indicado faz parte do processo de autenticação e é gerado após o utilizador fazer login. Quando este faz logout, o token guardado no armazenamento local do browser torna-se inválido e obriga o utilizador e fazer login novamente.

4.4.5 Guarda de autenticação

Esta funcionalidade do Angular, permite que determinados componentes só estejam ativos sob determinadas condições. O guarda de autenticação criado ()Figura 4.13) nesta secção, permite que os outros módulos não sejam carregados até que toda a informação seja carregada da base de dados.

```
{ path: 'task-manager', canActivate: [AuthGuard], component: TaskManagerComponent, children: [
```

Figura 4.13 - RouteGuard inserido no gestor de tarefas

A Figura 4.13 mostra um caso em que o guarda de autenticação está inserido, impossibilitando que o utilizador entre neste módulo mesmo colocando o caminho correto no navegador.

4.5 Centro da Aplicação

Este módulo, como função, tem o objetivo de iniciar o utilizador na aplicação, tendo apenas duas funções básicas: **login do utilizador** e **redireccionamento para outros módulos.**

Como tal, pode-se definir o estado do centro da aplicação em duas fases:

Apresentação ao utilizador

Nesta fase é apresentado um ecrã de boas-vindas (**Figura 4.14**) ao utilizador, requerendo que o mesmo faça login.

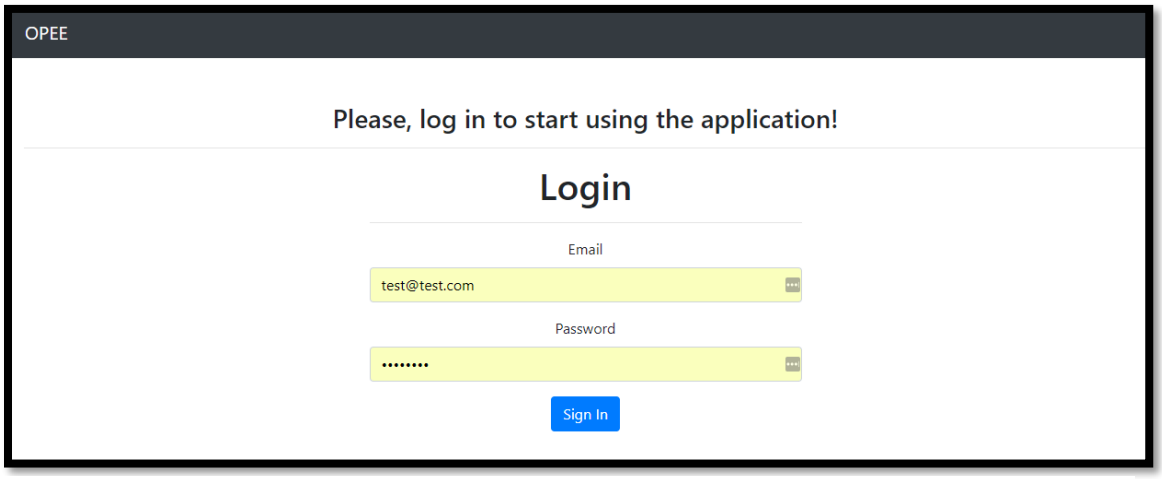

Figura 4.14 - Página inicial da aplicação pré-login

Como o utilizador ainda não fez login, não tem acesso a nenhum outro módulo no cabeçalho da aplicação.

Encaminhamento para os outros módulos

Posto o login bem-sucedido, é apresentada ao utilizador uma página com redireccionamentos para os outros módulos (Figura 4.15), assim como é preenchido so cabeçalho com a informação a que o utilizador tem acesso.

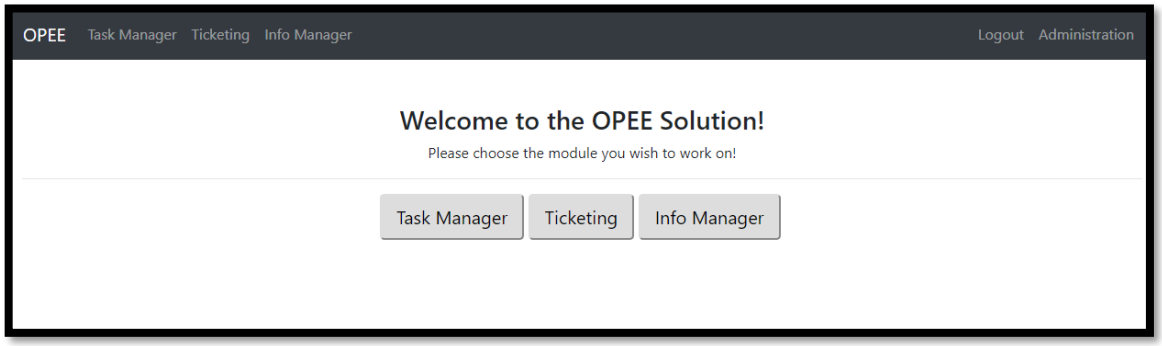

Figura 4.15 - Página inicial da aplicação pós-login

4.6 Gestor de Tarefas

No gestor de tarefas temos efetivamente o primeiro módulo funcional para os utilizadores. O objetivo principal é a gestão interna de tarefas entre colaboradores, gestão esta que é dividida em projetos, de forma a que os utilizadores só tenham acesso a determinada informação e não possam consultar tarefas que não lhes dizem respeito.

A primeira fase de implementação deste módulo passou por definir como é que a informação seria apresentada. Neste caso, optou-se por uma apresentação em lista e outra apresentação em quadro Kanban.

Na segunda fase, definiu-se o acesso aos projetos. É aqui que entra, pela primeira vez, o modelo de projetos falado anteriormente. Através da diretiva **ng-for** disponibilizada pelo angular, percorre-se o objeto que contém todos os projetos da aplicação (retirado da base de dados), e apresenta apenas os projetos a que o utilizador tem acesso, fazendo um cruzamento entro o campo **roleAcess** dos projetos e **role** do utilizador que se encontra ligado, recorrendo ao seguinte algoritmo (Figura 4.16).

```
hasAccess(project: Project) {
  if (project === null) {
    return false;
  } else if (project.roleAccess.indexOf(this.authService.loggedUser.role) > -1) {
    return true;
  } else {
    return false;
  }
}
```

Figura 4.16 - Algoritmo de apresentação de projetos do utilizador

O algoritmo da Figura 4.16 vai ser utilizado várias vezes ao longo da aplicação quando for necessário fazer esta triagem dos projetos que se querem apresentar.

Posta a apresentação dos projetos e a escolha do projeto pelo utilizador, são apresentadas as tarefas correspondentes ao projeto escolhido.

Esta fase do módulo divide-se em várias partes (Figura 3.4), sendo estas apresentadas seguidamente.

Criação e edição de tarefas

Para esta funcionalidade são utilizados os formulários reativos (funcionalidade do Angular), definindo os campos que se pretende que o utilizador preencha e definindo se os mesmos são obrigatórios ou não (Figura 4.17).

```
this.taskEditForm = new FormGroup({
  'name': new FormControl(this.task.name, Validators.required),
```

Figura 4.17 – Exemplo de obrigatoriedade de validação

Tanto para a criação como para a edição de tarefas, utiliza-se o mesmo módulo (Figura 4.18), fazendo apenas a validação se o utilizador está em modo de edição.

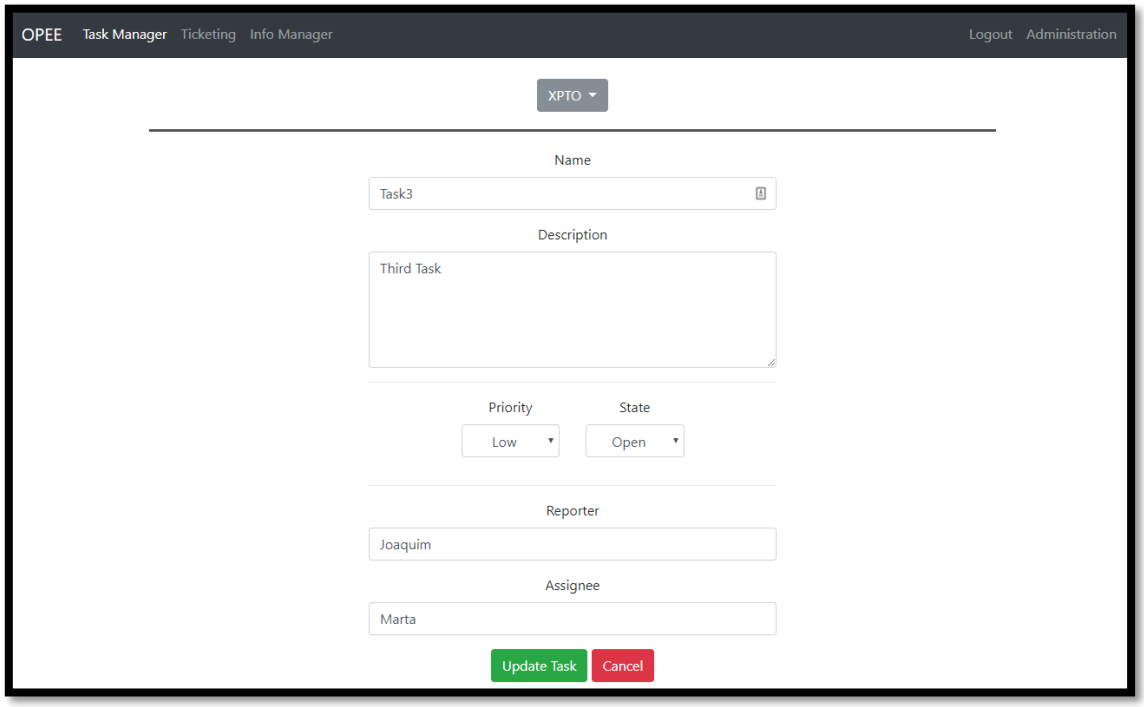

Figura 4.18 - Página de edição/criação de tarefa

s

Quando o formulário é submetido, guarda-se a tarefa na base de dados, recorrendo ao método da Figura 4.19.

```
// store tasks in backend
this.dataStorageService.storeTasks()
.subscribe(
  (response: HttpEvent<Object>) => {
    console.log(response);
  }
);
```

Figura 4.19 - Método para gravação de dados na BD

Vista das tarefas por listagem

Uma vez criada a tarefa, o utilizador por recorrer a este modo de visualização de conteúdo, que é apenas uma tabela que contém todas as tarefas do projeto, listando-as recorrendo à diretiva **ng-if** (Figura 4.20).

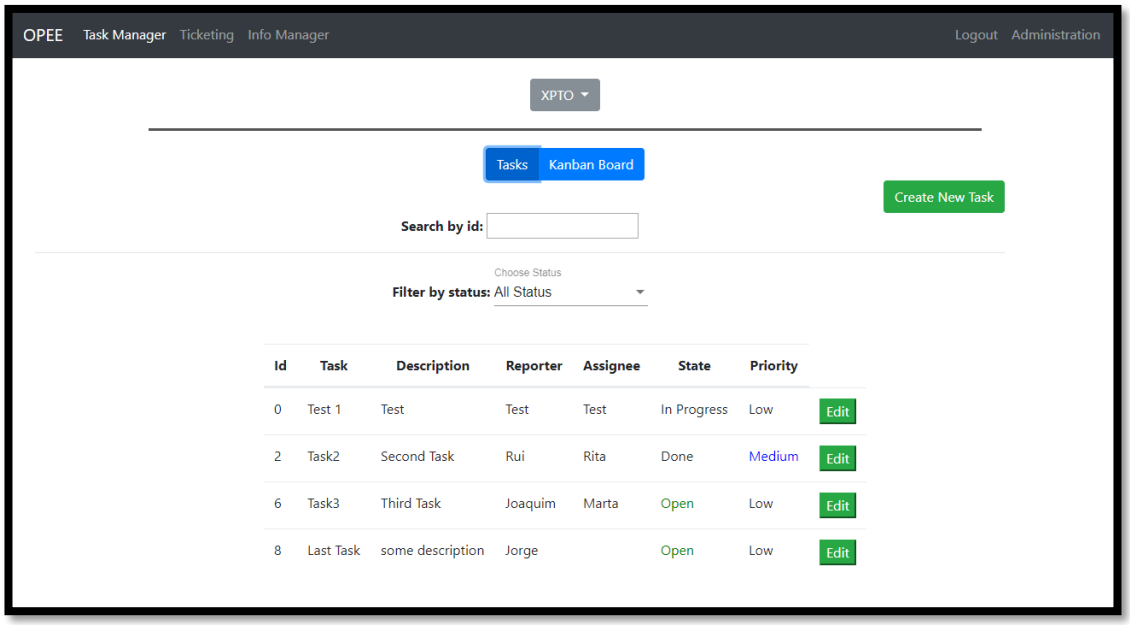

Figura 4.20 - Visualização em lista

Esta página contém ainda duas funções importantes: **filtro por id** (Figura 4.21) e **filtro por estado** (Figura 4.22).

```
// id filter
if (this.filteredID !== '') {
  if (task.id === Number(this.filteredID)) {
    return true;
  } else {
    return false;
  }
}
```

Figura 4.21 - Filtro de tarefas por id

```
// status filter
if (this.filteredStatus === 'All Status') {
  return true;
} else if (this.filteredStatus === 'Open') {
  if (task.state === 'Open') {
    return true;
  }
  return false;
} else if (this.filteredStatus === 'In Progress') {
  if (task.state === 'In Progress') {
    return true;
  }
  return false;
} else if (this.filteredStatus === 'Done') {
  if (task.state === 'Done') {
    return true;
  }
  return false;
} else if (this.filteredStatus === 'Closed') {
  if (task.state === 'Closed') {
    return true;
  }
  return false;
}
```

Figura 4.22 - Filtro de tarefas por estado

Nas figuras Figura 4.21 e Figura 4.22 pode-se ver uma lógica semelhante, uma vez que o resultado booleano é colocado na condição de mostragem de tarefas, correspondendo cada resultado apenas a uma tarefa. Se for verdadeiro a tarefa é mostrada, se for falso, não o é.

Vista de tarefas por quadro Kanban

A vista neste modo tem como objetivo visualizar as tarefas por estado, para quase instantaneamente, o utilizador conseguir visualizar as tarefas nos estados mais críticos.

O modo de apresentação das tarefas é semelhante ao método da listagem anterior e aqui dá-se mais importância ao aspeto visual de forma a apelar mais ao utilizador (Figura 4.23).

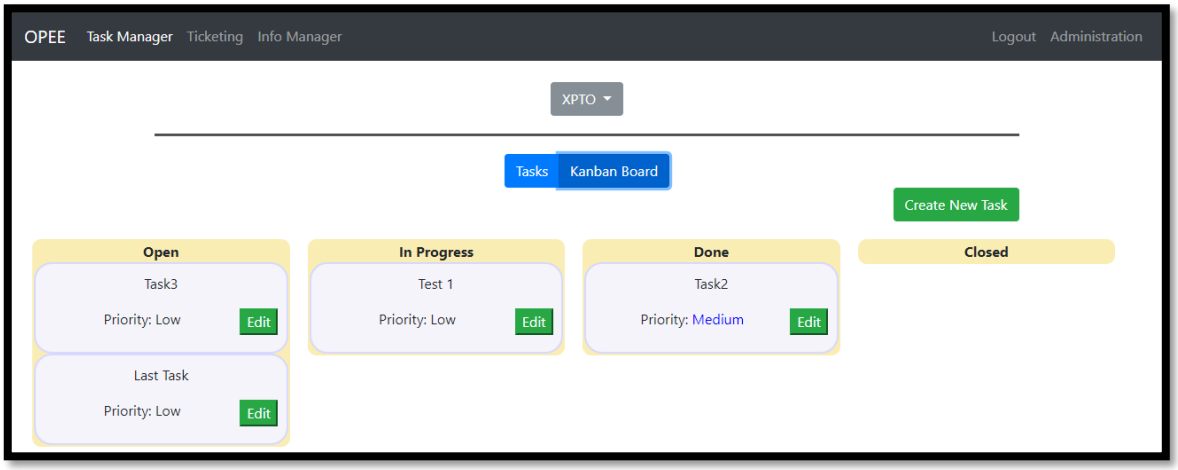

Figura 4.23 - Visualização em Kanban-Board

Vista individual de tarefas

A vista individual de tarefas (Figura 4.24) é outro componente específico para este efeito. Aqui vai-se recorrer à tarefa escolhida do utilizador e apresentar os campos. Também há a possibilidade de editar a tarefa e de **apagar uma tarefa.**

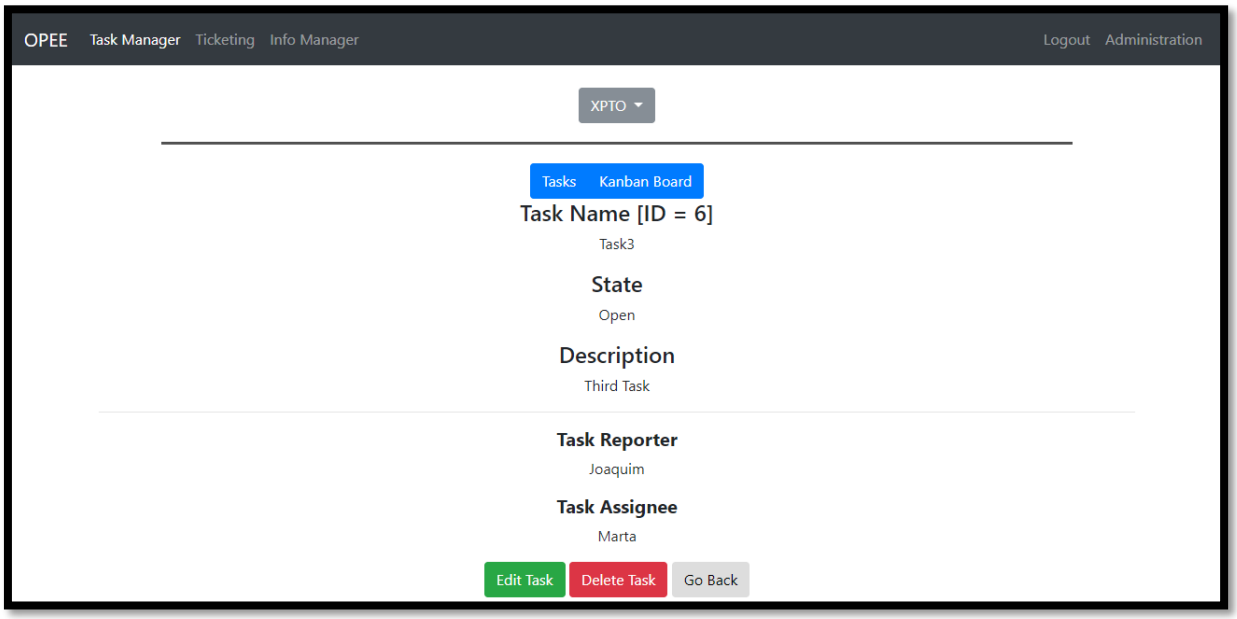

Figura 4.24 - Página de visualização de tarefa

O algoritmo para apagar a tarefa foi apresentado em Figura 4.7 - Método para remover um projeto na BD e aqui (Figura 4.25) apenas se faz a chamada para o mesmo.

```
onDeleteTask() {
  this.dataStorageService.removeTask(this.taskSelected);
  this.router.navigate(['../tasks'], {relativeTo: this.route});
}
```

Figura 4.25 - Chamada do método de remoção de tarefa

4.7 Ticketing

O propósito de construção deste módulo é a comunicação entre projetos. Toda a sua interface e funcionalidades têm de seguir este objetivo.

Como tal, o método de comunicação é muito parecido com o gestor de tarefas, mas a interface e comunicação têm de ser ajustados de forma a realçar a informação relevante apresentada para os outros projetos, como exemplificado na Figura 4.26.

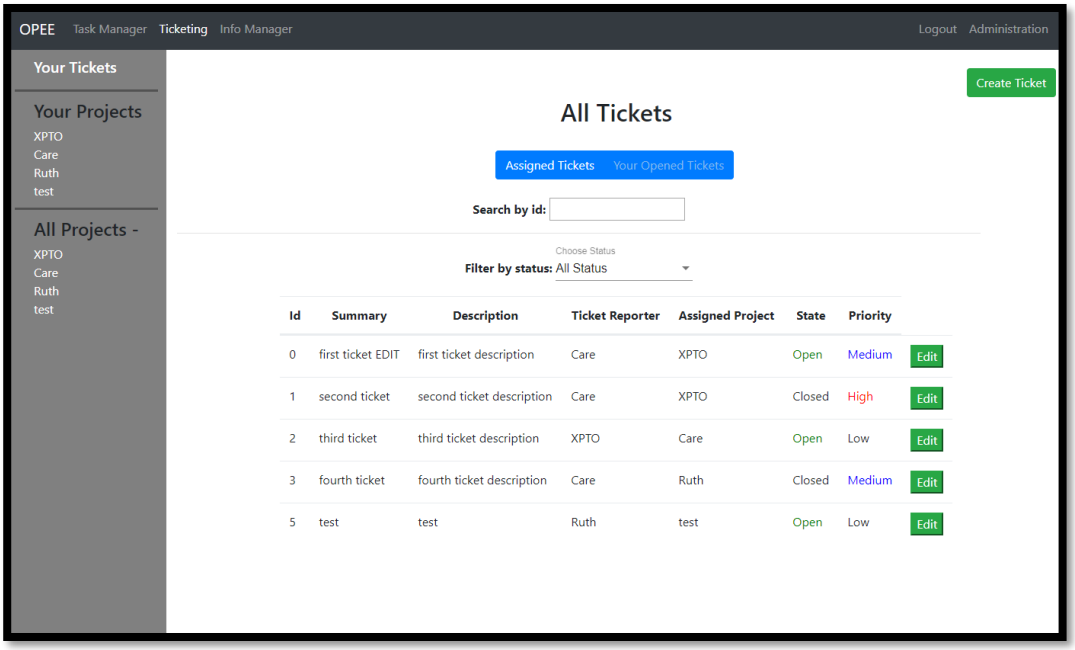

Figura 4.26 - Página inicial do módulo de ticketing

O modelo de apresentação de ticketing está dividido em três secções e duas subsecções. Relativamente às secções, apresentam-se por:

- **Tickets a que o utilizador tem acesso –** Nesta secção o utilizador poderá consultar todos os tickets que têm acesso sem discriminização do projeto;
- **Tickets a que o utilizador tem acesso discriminados por projeto –** Aqui o utilizador escolherá o projeto que pretende observar e serão apresentados os tickets correspondentes ao mesmo;

• **Todos os tickets discriminados por projeto** – Como também, em certas situações, será necessário consultar tickets de projetos externos, apresenta-se esta secção também. O utilizador não poderá editar tickets de projetos externos, ao contrário do que acontece aos seus próprios tickets.

As subsecções, referentes apenas às primeiras duas secções, dividem-se em:

- **Tickets abertos para o projeto –** Listam-se os tickets que foram abertos com o projeto destinatário igual ao projeto atual;
- **Tickets abertos pelo projeto** – Listam-se os tickets que foram abertos pelo projeto atual com outro projeto destinatário.

Posta esta definição de apresentação dos tickets, passou-se à segunda fase de implementação, a **visualização das tickets**.

Visualização de Tickets

Nesta fase, à semelhança do gestor de tarefas, percorre-se a lista das tarefas tendo em consideração três condições.

O processo de escolha do tipo de visualização dá-se pelo fluxograma representado na Figura 4.27.

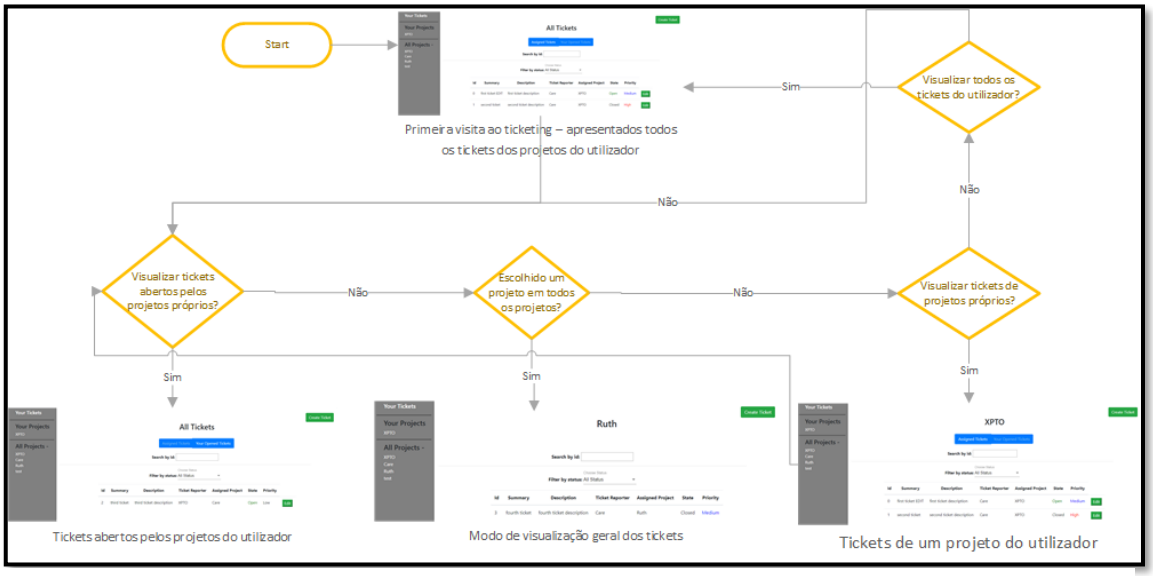

Figura 4.27 - Fluxograma de escolha de visualização

Escolhendo o utilizador a **visualização dos tickets a que tem acesso por projeto**, apresentam-se apenas os tickets cujo projeto destinatário é igual ao projeto escolhido.

Caso o utilizador pretenda observar **todos os tickets que foram atribuídos a projetos que ele tem acesso**, utiliza-se o método da Figura 4.28.

```
hasAccess(roleAccess: string[]) {
  if (roleAccess.indexOf(this.loggedUser.role) > -1) {
    return true;
  } else {
    return false;
  }
}
```

Figura 4.28 - Método de acesso aos tickets por projeto

Aqui apenas se verifica se o role que o utilizador tem, isto é, se tem a função de Administrador, Recursos Humanos, etc… e confirma se essa função está inserida nas funções do projeto (campo roleAccess de cada projeto). Se a função estiver inserida, o ticket é apresentado.

Se o utilizador pretende visualizar **todos os tickets por projeto,** também se usa o método de comparação do projeto destinatário com o projeto escolhido, mas retira-se a possibilidade de visualizar os tickets abertos pelo projeto selecionado.

As restantes funcionalidades presentes nestes módulos são a **criação e edição de tickets** e também a possibilidade **de filtrar a apresentação dos tickets por id ou status.** O método para a filtragem de conteúdo é o mesmo utilizado nas Figura 4.21 e Figura 4.22.

Criação e Edição de tickets

Esta funcionalidade (Figura 4.29) também tira proveito dos formulários reativos proporcionados pelo Angular e também são definidos os campos que são obrigatórios ao preenchimento.

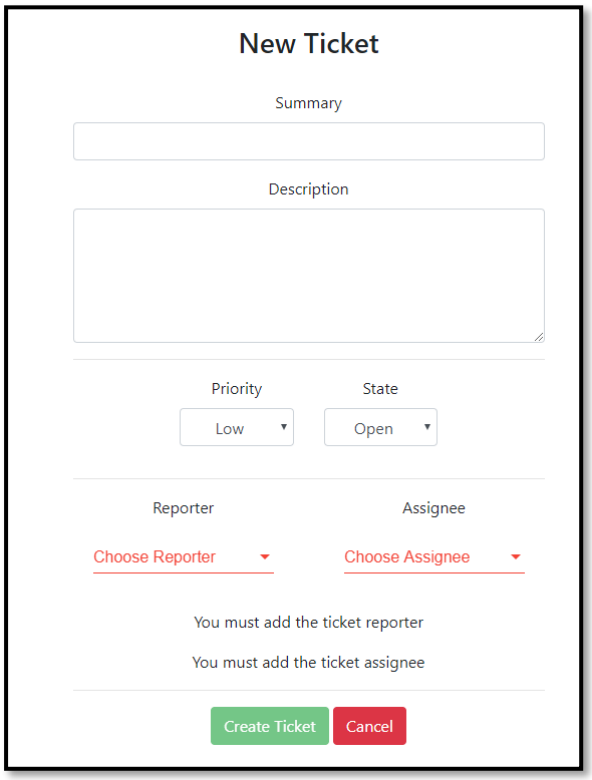

Figura 4.29 - Criação de um ticket

Nesta página, uma vez que o utilizador pode estar inserido em vários projetos, é escolhido o projeto original da abertura do ticket e o projeto destinatário para a resolução do mesmo.

Relativamente à edição dos tickets, estes podem ser editados por utilizadores do projeto original e por utilizadores do projeto destinatário.

Já a vista individual de um ticket é semelhante ao de uma tarefa, mas não contém a capacidade de remover os tickets. Se for necessário tornar inválida a informação de um ticket, tem que se colocá-lo no estado "Closed".

4.8 Gestor de Informação

O módulo do gestor de informação tem o objetivo simples de guardar informação. Esta informação é dividida por **projetos** e por **secções dos projetos**.

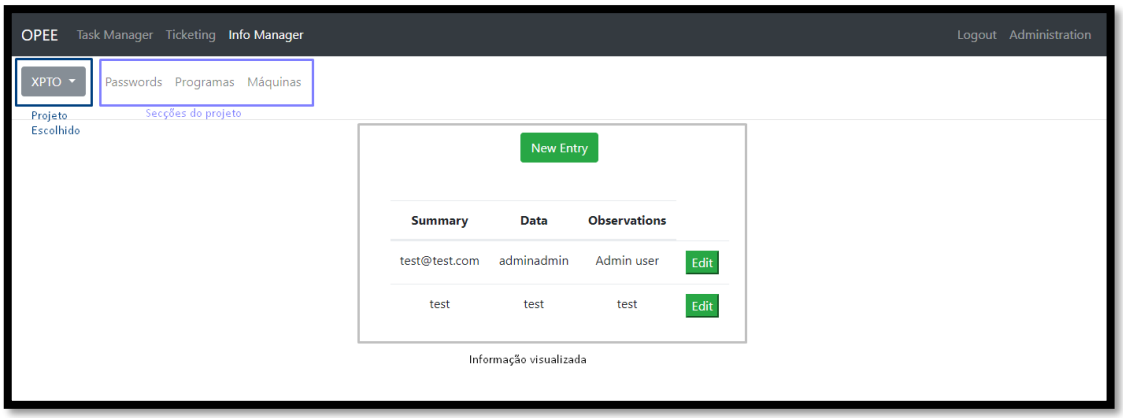

Figura 4.30 - Modelo de apresentação da informação

O modelo de construção da apresentação de informação neste módulo é dividido em três blocos (Figura 4.30).

No bloco do projeto escolhido, o utilizador terá acesso aos projetos com base no seu tipo de utilizador (Figura 4.31). É criado um objeto apenas com os projetos do utilizador e é apresentado no botão de escolho.

```
getProjectsByRole(role: string) {
  const projects = this.projects.slice();
  const remIndex: number[] = [];

  // gets the index of the items to be removed
  for (const item of projects) {
    if (item === null) {
      remIndex.push(projects.indexOf(item));
    } else if (!item.roleAccess.includes(role)) {
      remIndex.push(projects.indexOf(item));
    }
  }

  // removes the items
  for (let i = remIndex.length - 1; i >= 0; i--) {
    projects.splice(remIndex[i], 1);
  }

  return projects;
}
```

Figura 4.31 - Algoritmo de filtragem dos projetos do utilizador

No bloco das secções do projeto é utilizada a diretiva **ng-for** para apresentar todas estas, uma vez que cada projeto tem guardado em **"infoSections"** todas as secções que o mesmo contém.

Criação, Edição e Remoção de Tarefas

A capacidade de criar, editar e remover tarefas também está presente neste módulo. Apenas se introduziu a obrigatoriedade de inserção de campo "**summary"** uma vez que nem todo o tipo de informação é igual e cada tarefa pode ou não necessitar de todos os campos presentes no modelo de informação.

A capacidade de editar tickets está tanto disponível na listagem de cada ticket como na vista individual de cada um e a capacidade de o remover apenas está contido na vista individual.

4.9 Consola de Administração

O módulo final é a consola de administração e neste não é suposto existir um fluxo de organização de trabalho, mas sim definir certos objetos nos módulos, objetos estes que só os administradores têm acesso.

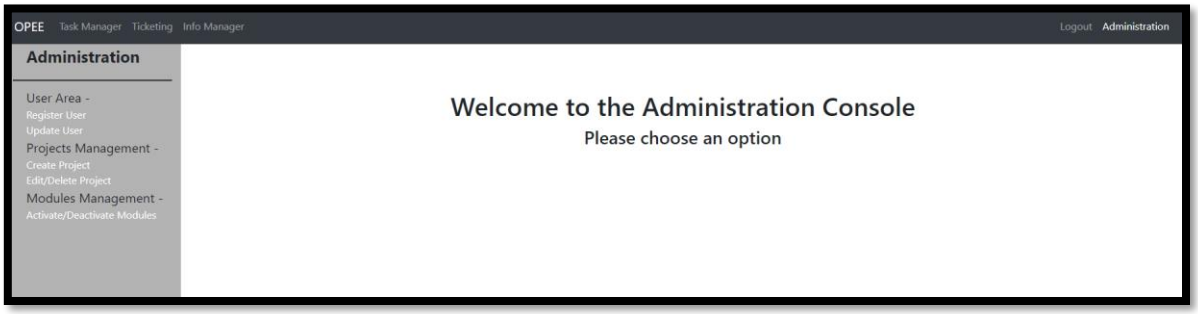

Figura 4.32 - Ecrã inicial do painel de Administração

As áreas editáveis da consola estão divididas em três secções (Figura 4.32):

- **Área de utilizador –** Secção de registo e atualização de utilizadores;
- **Gestão de projetos –** Secção de criação, edição e remoção de projetos;
- **Gestão de módulos –** Secção para controlo dos módulos ativos.

Optou-se apenas por esta divisão em três secções uma vez que as áreas editáveis pelos administradores em todos os módulos estão contidas no modelo de projeto, áreas estas:

- **Utilizadores com acesso –** Utilizadores que terão acesso ao projeto (áreas de trabalho);
- **Secções de informação –** Secções de informação no gestor de informação;
- **Administradores do projeto –** Utilizadores com acesso administrativo aos projetos.

A definição dos métodos para a criação e alteração de utilizadores foram definidas em 4.4, portanto só se definirá seguidamente o funcionamento da gestão de projetos e gestão de módulos.

Criação de Projetos

Esta funcionalidade (Figura 4.33) é a raiz de utilização de todos os módulos implementados. A criação de um projeto é o que vai permitir a inserção de dados em todos os outros módulos e apenas um administrador da aplicação poderá criar projetos.

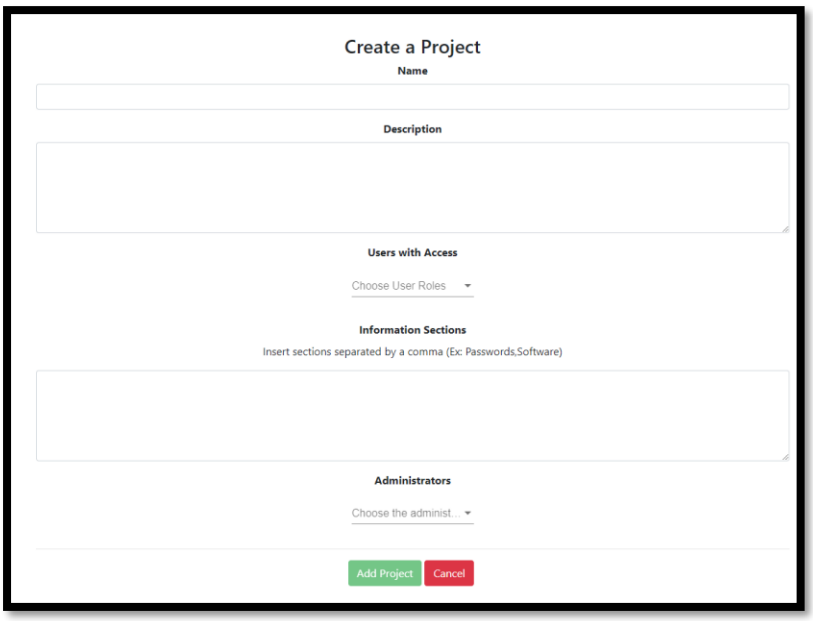

Figura 4.33 - Módulo para criação de projeto

O módulo para a criação de um projeto define também um formulário reativo e define o "**Name**", "**Description**" e **"Users with Access"** como campos obrigatórios para se iniciar um projeto. Caso não haja secções de informação e caso não haja administradores num projeto não são problemas que invalidem o bom funcionamento da aplicação.

Edição e Remoção de projetos

A edição dos projetos conta, inicialmente, com a listagem de todos os projetos a que o utilizador tem acesso (Figura 4.34). Caso seja um administrador da aplicação, terá acesso a todos os projetos, caso seja administrador de projetos, apenas terá acesso aos projetos que lhe dizem respeito.

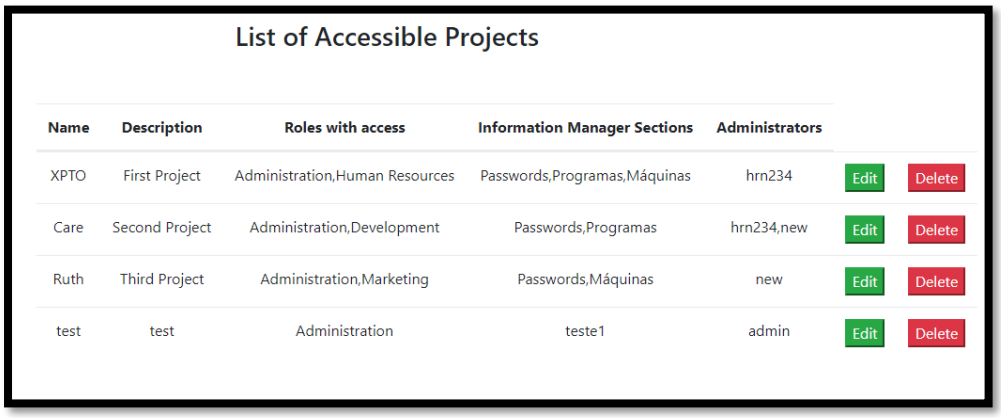

Figura 4.34 - Lista de projetos existentes

Tanto os administradores da aplicação como os administradores de projeto podem apagar totalmente os seus dados.

Gestão de Módulos

Esta área tem apenas como função ativar e desativar a visualização de módulos criados (Figura 4.35). Recorrendo aos **serviços** (funcionalidade do Angular), é possível atualizar em tempo real o cabeçalho da aplicação e eliminar/inserir o redireccionamento para o módulo ativado/desativado.

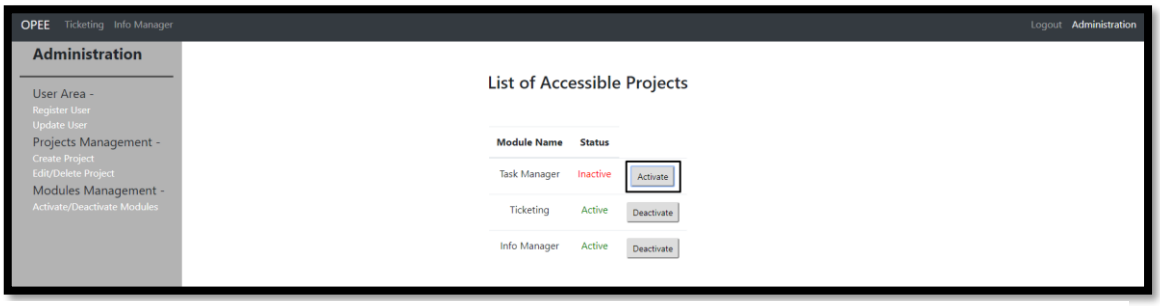

Figura 4.35 - Vista de projetos acessíveis

Carregando no botão realçado na Figura 4.35 recorre-se ao serviço dos módulos e atualiza-se o objeto dos módulos que foi carregado no início da aplicação. A informação é enviada para a base de dados e da próxima vez que o utilizador entrar na aplicação só os módulos ativos é que terão rotas de acesso.

5

5 Validação

A solução conceptual apresentada tem como objetivo agilizar a realização de tarefas secundárias dos trabalhadores da forma mais rápida e eficiente possível. É com este intuito que, para validar a solução, pretende-se perceber se as tarefas são fáceis e rápidas de se realizar e que a informação disponível para os utilizadores é clara.

Para efeitos de validação define-se então dois critérios:

- **Tempo de realização da tarefa –** Tempo que o utilizador demorou a realizar a tarefa definida;
- **Compreensão da tarefa –** Compreensão das tarefas por parte do utilizador.

Para perceber efetivamente se uma tarefa é fácil de realizar, optou-se por não contextualizar o uso da aplicação aos utilizadores e apenas pedir a realização das seguintes tarefas em cada módulo:

Gestor de Tarefas

- **T1 -** Procurar uma tarefa com o id igual a 2 no projeto XPTO;
- **T2 -** Trocar o estado dessa mesma tarefa para Open;
- **T3 –** Filtrar as tarefas pelo estado "Open**";**
- **T4 –** Criar uma tarefa tendo o utilizador como relator e o administrador como destinatário.

Ticketing

- **T5 -**Visualizar apenas os tickets do projeto Ruth (projeto de teste criado)0
- **T6** - Criar um ticket do projeto Ruth para o projeto XPTO;
- **T7** - Visualizar os tickets abertos pelo projeto Ruth;
- **T8** - Visualizar todos os tickets e filtrar pelo estado "Open";
- **T9** - Trocar a prioridade do ticket com id igual a 3 para "High".

Gestor de Informação

- **T10 -** Visualizar as máquinas do projeto XPTO;
- **T11 -** Adicionar nova máquina definindo um sumário e dados.

A população utilizada foram oito pessoas dos 22 aos 62 anos, com habilitações equivalentes do 12º ano a licentura e com uma boa compreensão de inglês. A idade definida é a idade normal de um trabalhador ativo para obter uma maior fiabilidade de resultados.

O tempo médio de realização das tarefas propostas por parte dos utilizadores está representado na Figura 5.1, obtendo uma média total de tempo de resposta por tarefa de 22,2 segundos.

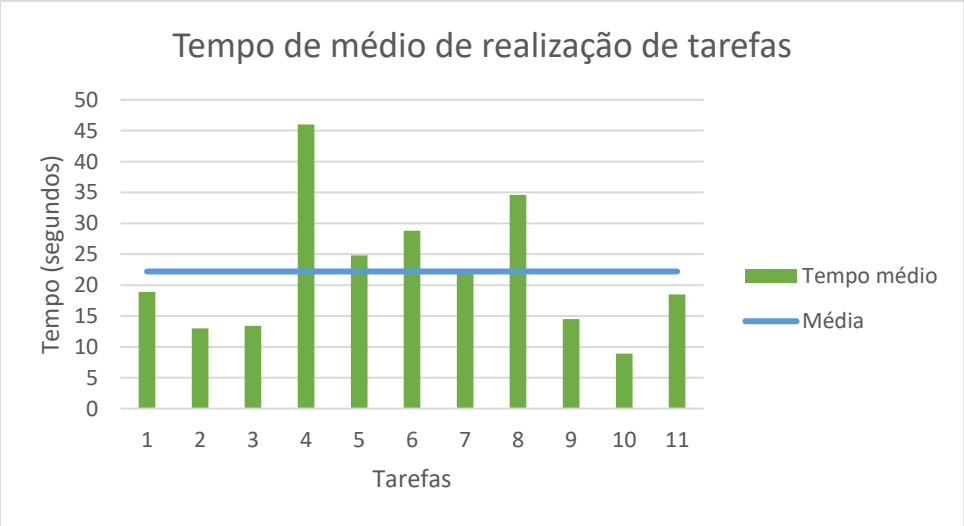

Figura 5.1 - Tempo médio de realização de tarefas

Observando o resultado pode-se constantar que efetivamente a aplicação é intuitiva e proporciona uma resposta rápida aos utilizadores, como pretendido.

Para confirmar se existiu um elevado nível de compreensão das tarefas, observa-se o gráfico da Figura 5.2 onde a média de compreensão de tarefas ficou situada nos 8,5 valores

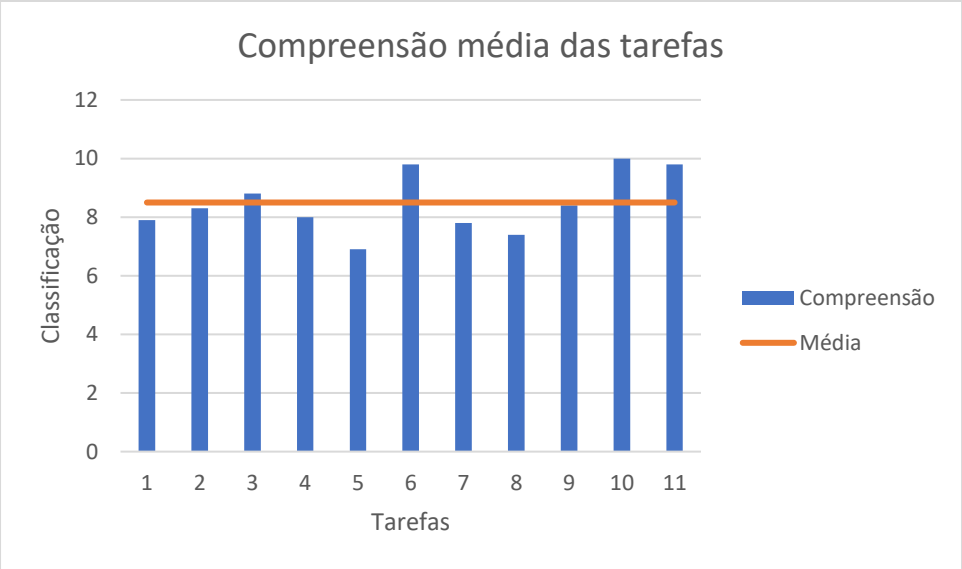

Figura 5.2 - Compreensão média das tarefas (classicação de 0 a 10)

Uma vez que se conseguiu obter um baixo nível de tempo de realização de tarefas e uma elevada compreensão das mesmas por parte dos utilizadores, pode-se concluir que os requisitos definidos em 3.2 foram cumpridos.

6

6 Conclusões

O maior desafio do trabalho proposto foi conseguir criar uma solução conceptual que, ao mesmo tempo que produzia uma usabilidade real, conseguia criar um modelo de construção para futuras ferramentas, isto é, criar uma espécie de fórmula para criar mais soluções com um modelo de interface simples e intuitivo.

A solução proposta consegue preencher estes dois campos porque, ao mesmo tempo que funciona num modelo modular, consegue proporcionar uma experiência rápida e útil para os seus utilizadores. O facto da sua construção ser baseada numa das *frameworks* mais atuais e mais utilizadas no mercado também facilita a longevidade deste modelo modular.

Um dos maiores problemas encontrados foi a capacidade de criar interfaces bonitas e agradáveis ao utilizador e, como tal, o aspeto funcional foi tido como prioritário no desenvolvimento da aplicação. Outra das dificuldades foi a aprendizagem de raiz da *framework* e a quantidade de dependências que o desenvolvimento web com outras bibliotecas/ferramentas.

A plataforma de *back-end* utilizada, apesar de se encaixar bem no contexto desta solução, também não é muito escalável para projetos maiores e, para uma solução mais abrangente, seria necessário desenvolver uma camada de *back-end* de raiz e mais coesa.

O objetivo desta proposta nunca foi resolver todos os problemas existentes no mercado, mas sim proporcionar uma janela para a resolução dos mesmos. O

que se propõe é que se criem ferramentas escaláveis e com capacidade de adição de funcionalidades de forma a que as que já existem não se tornem obsoletas ou não tornem a aplicação mais difícil de se utilizar.

6.1 Sugestões e trabalhos futuros

Para distribuir e efetivamente implementar esta solução numa empresa seria necessário desenvolver a camada de *back-end* customizada para este tipo de projeto, assim como melhorar a interface geral do programa e torna-lo mais apelativo aos utilizadores gerais.

A tradução para português e outras linguagens também é um aspeto importante, uma vez que alguns dos utilizadores que testaram a aplicação não sabiam inglês e tornou mais difícil a sua navegação na mesma.

A possibilidade de adicionar, no gestor de tarefas, a capacidade de tornar umas tarefas dependentes das outras, assim como a data de introdução e finalização da tarefa seria algo que tornaria a tarefa mais informativa, embora também tornasse a sua abertura num processo um pouco mais complexo, sendo que o contrário era o pretendido nesta solução.

Uma vez que a solução foi apenas otimizada para computadores, seria interessante também otimizá-la para dispositivos móveis de acordo com o contexto de utilização de dispositivos atual da sociedade.

Para tornar a solução mais apelativa há sempre inúmeras funcionalidades que se podem acrescentar ou melhorar, sendo que estas são apenas algumas das soluções mais imediatas que se poderiam implementar.

7

7 Bibliografia

Beck, K., & Andres, C. (2004). Extreme Programming Explained.

Davenport, T., & Dyché, J. (2013). Big Data In Big Companies. *Journal of Parallel and Distributed Computing*, *74*(7), 20–21.

Facebook. (n.d.-a). Introducing JSX - React. Retrieved January 26, 2018, from https://reactjs.org/docs/introducing-jsx.html

Facebook. (n.d.-b). Rendering Elements - React. Retrieved January 26, 2018, from https://reactjs.org/docs/rendering-elements.html

Glover, M., Hillsberg, A., Trello, J., Renford, R., Adelman, D., Sibbet, C., & Bead, E. (n.d.-a). Jira Reviews: Software Overview, Pricing and Features. Retrieved January 24, 2018, from https://reviews.financesonline.com/p/jira/

Glover, M., Hillsberg, A., Trello, J., Renford, R., Adelman, D., Sibbet, C., & Bead, E. (n.d.-b). Software Reviews & Business Advice | FinancesOnline.com. Retrieved February 24, 2018, from https://reviews.financesonline.com/

Google. (n.d.). Angular - Architecture Overview. Retrieved January 26, 2018, from https://angular.io/guide/architecture

Graziotin, D., & Abrahamsson, P. (2013). Product-Focused Software Process Improvement, *7983*(Profes), 334–337. https://doi.org/10.1007/978-3-642-39259-7

Pressman, R. S. (2011). *Engenharia de Software*. *Engenharia de Software*. https://doi.org/9788563308337

Rainardi, V. (2007). *Building a Data Warehouse: With Examples in SQL Server*. *Informatik-Spektrum* (Vol. 20).

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum*. *cdswebcernch*. https://doi.org/10.1109/2.947100

Soares, M. D. S. (2003). Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. *INFOCOMP Journal of Computer Science*, *27*(2), 6. https://doi.org/10.4067/S0718-34292009000200002

You, E. (n.d.). Introduction — Vue.js. Retrieved January 26, 2018, from https://vuejs.org/v2/guide/